Braun
Dittrich
Schramm

# ATARI® ST

# Floppy und Harddisk

EIN DATA BECKER BUCH

Braun
Dittrich
Schramm

# ATARI® ST
# Floppy und Harddisk

EIN DATA BECKER BUCH



ISBN 3-89011-132-7

# Inhaltsverzeichnis

# 1. Einleitung

Die Rechner der ATARI ST-Serie sind mit ihrem schnellen 16/32-Bit-Prozessor und ihrer hohen Speicherkapazität für professionelle Anwendungen wie geschaffen. Doch wichtiger als der interne Speicher ist die Möglichkeit der externen Datenspeicherung. Die hierfür verwendeten Floppy-Disks und die Harddisk sind sehr interessante und vielseitige Speichermedien, die mehr können als man im Handbuch findet.

Für die optimale Anwendung eines solchen Computer-Systems ist es wichtig, die Fähigkeiten der Komponenten zu überschauen. Hierfür ist dieses Buch ideal geeignet. Es gibt zunächst einen Einblick in die Welt der Massenspeicher und beschreibt die Vorgehensweise bei der Programmierung von Anwendungsprogrammen. Dann wird immer tiefer in die Geheimnisse der ATARI-Floppys sowie der Harddisk und auch RAM-Disks eingegangen.

All diese Kenntnisse der Soft- und Hardware versetzen Sie in die Lage, die interessantesten Dinge mit diesen Speichern anzufangen. Sie können die Kapazität der Disketten erhöhen, einen eigenen Kopierschutz für Ihre Programme entwerfen, eine eigene RAM-Disk nach Ihren Bedürfnissen erstellen und mit den in diesem Buch enthaltenen Beispiel- und Hilfsprogrammen Ihre Programme wesentlich schneller und effektiver auf Disketten und/oder eine Harddisk zugreifen lassen.

Weiterhin finden Sie sehr nützliche Programme, die z. B. ein komlettes Inhaltsverzeichnis inclusive Ordnerinhalte übersichtlich auf dem Drucker ausgeben oder Disketten bzw. die Festplatte analysieren.

Als besonderes Bonbon finden Sie ebenfalls einen kompletten und sehr vielseitigen Diskettenmonitor, ein Programm, mit dem Sie direkt auf die Disketten zugreifen und somit alle erworbenen Kenntnisse direkt anwenden können. So können gelöschte Dateien wieder gerettet, Fremd-Disketten gelesen werden usw.

Sie werden in diesem Buch einige Dinge finden, die in keinem Handbuch auftauchen. Diese Kommandos oder Zusammenhänge sind in langwieriger Kleinarbeit zusammengetragen worden, so daß das Buch etwas später, aber auch mit sehr vielen Informationen erschienen ist. Sie werden feststellen, daß die Floppies und Harddisk wesentlich mehr kann, als Sie gedacht hatten!

Wir hoffen, daß Sie mit Hilfe dieses Buches alle Fragen bezüglich der Massenspeicher umfassend beantwortet bekommen. Und nun, viel Spaß mit der Lektüre des Buches!

# 2. Files, Programme und Dateien

Die drei Begriffe der Kapitelüberschrift stehen im Grunde für einen: Computerdaten jeglicher Art auf einem externen Speichermedium. Trotz größer werdender Hauptspeicher der Rechner, der ATARI ST+ hat immerhin 1 Megabyte RAM, müssen vom Rechner momentan nicht benötigte Daten, z.b. das Textverarbeitungs-Programm, die Einwohner der Stadt Köln oder die Kakaopreise der letzten 50 Jahre, auf einem externen Speichermedium untergebracht werden, da sie sonst ja beim Ausschalten des Rechners verloren gingen.

Als externes Speichermedium benutzt man heute Magnetbänder, Disketten, Harddisks und neuerdings auch CD-ROMs. Bei allen diesen sogenannten Massenspeichern werden die Daten erst in irgendeiner Form auf das Medium übertragen und anschließend durch eine geeignete Leseelektronik wieder in den Hauptspeicher eingelesen. Unabhängig von der Art des Massenspeichers bezeichnet man die Gesamtheit der gespeicherten Daten unter einem Namen auf der Diskette (o.ä.) als Datei oder File (engl. Akte).

Ob als Daten nun Adressen, Brief- und Programmtexte oder ausführbarer Programmcode gespeichert wird, ist für den Anwender nebensächlich, für den Rechner jedoch von fundmentaler Bedeutung.

Bei der Speicherung von Programmcode darf kein Trennzeichen zwischen einzelnen Daten vorhanden sein. Anders sieht dies bei abgespeicherten Texten aus, wo zwischen den einzelnen Sätzen oft entsprechende Trennzeichen, z.B. einfach der der Return-Taste entsprechende Code, eingesetzt werden.

Der Anwender erkennt den Typ einer Diskettendatei meist schon an den drei zusätzlichen Buchstaben des Namens, dem Extender. Das Betriebssystem des ATARI ST unterscheidet Programme oder Dateien nur an diesem Extender. Benennt man also ein Programm (.PRG) um und hängt statt PRG den Anhang DAT an, so wird nach Anklicken dieses Programmes nur das bekannte

Auswahlfenster zum Ausdruck oder Ansehen der Datei erscheinen.

Die vom ATARI ST direkt unterschiedenen Extender sind:

*.PRG* kennzeichnet ein lauffähiges Maschinen-Programm, welches mit GEM-Unterstützung laufen kann.

*.TOS* bedeutet ebenfalls, daß dies ein Maschinen-Programm ist, beim Ablauf dieses Programmes ist das GEM jedoch abgeschaltet.

*.TTP* gleicht .TOS, vor dem Aufruf des Programmes selbst erscheint jedoch ein kleines Fenster, in dem man Parameter (z.B. Dateinamen für Editoren) eingeben kann.

*.ACC* sind spezielle Maschinenprogramme, die nach dem Einschalten des Rechners geladen werden. Diese Programme bleiben ständig im Speicher und sind als Accessories aus dem DESK-Menü des Desktop aufrufbar.

*.INF* ist als DESKTOP.INF für das Desktop wichtig. Hierin sind die Informationen über die Positionen und Größen der Fenster, die im Kontrollfeld eingestellten Werte usw. eingetragen. Diese Datei wird durch Anwahl des Menüpunktes 'Arbeit sichern' im EXTRAS-Menü erstellt.

Die anderen Dateien wie z.B. BASIC-Programme mit dem Extender .BAS sind zwar standardmäßig mit diesem Extender ausgestattet, sind jedoch für das Betriebssystem des ST uninteressant. Außerdem kann man auch z.B. eine .TXT-Datei in den BASIC-Interpreter laden, wenn sie den Text eines BASIC-Programmes enthält. Die weiteren Extender sind also nicht entscheidend, sind jedoch für den Anwender eine große Hilfe, um seine Dateien zu unterscheiden.

Die eigentlichen Unterschiede zwischen den einzelnen Dateitypen liegen in dem inneren Aufbau der Datei selbst. Die meisten Hochsprachen unterscheiden bei der Dateibehandlung zwi-

schen verschieden Dateiformen, z.B. mit oder ohne Trennzeichen zwischen Texten oder Zahlen, spezielle Textarten usw. Betrachten wir nun erst einmal die reinen Daten-Files, die nur Texte und Zahlen enthalten. Hier gibt es verschiedene Methoden, bestimmte Daten aus der Datei herauszufinden und zu bearbeiten.

Die Geschwindigkeit des Zugriffs auf bestimmte Daten von Diskette oder Festplatte hängt hauptsächlich von der "Intelligenz" des Dateiverwaltungssystems ab. Dies läßt sich am besten an einem konkreten Beispiel verdeutlichen:

In einer Datei seien z.B. die Adressen aller Einwohner Kölns gespeichert. Die gesamten zu einer Adresse gehörende Information wie Name, Vorname, Postleitzahl, Wohnort, Straße und Hausnummer bezeichnet man als Datensatz, die einzelne Information wie z.B. der Name ist ein Datenfeld des Datensatzes. Die einfachste Form der Dateiverwaltung ist die einer sequentiellen Datei, bei der die Daten einfach der Reihe nach hintereinander geschrieben werden. Das Programm, welches solche Daten aus der Datei liest, muß das Ende eines Datensatzes selbst erkennen, da ein Trennkennzeichen nur zwischen den einzelnen Datenfeldern eingefügt wird.

Jeder Datensatz hat meist eine unterschiedliche Länge. Möchte man nun z.B. auf den 10. Datensatz zugreifen, muß man die Datei von Anfang an bis zum 10. Datensatz durchlesen. Nun ist dieses Verfahren für kleinere Datensätze noch akzeptabel, doch stellen Sie sich einmal vor man sucht in einer solchen sequentiellen Datei mit den Adressen aller Bundesbürger nach der von Harry Hirsch aus Buxtehude.

Müssen große Datenmengen verwaltet werden, arbeitet man meist mit festen Datensatzlängen und sogenannten RANDOM-ACCESS Dateien (random= beliebig, access= Zugriff) mit wahlfreiem Zugriff. In diesen Dateien steht jedem Datenfeld eines Datensatzes eine bestimmte, vorher festgelegte Größe zur Verfügung, z.B. 13 Zeichen für den Namen, 13 für den Vornamen, 4 für die Postleitzahl, 15 für den Ortsnamen, 15 für den Straßennamen und 4 für die Hausnummer, zusammen 64 Zeichen pro Datensatz.

Möchte man nun auf den 10. Datensatz zugreifen, kann man den Anfang des 10. Datensatzes relativ zum Anfang der Datei durch einfaches Multiplizieren berechnen. Man braucht dann nur am 10*64=640sten Byte der Datei zu lesen beginnen und hat direkt die gewünschten Daten parat. Die Rechnung gilt natürlich nur, wenn es auch einen nullten Datensatz gibt.

Dieser Trick funktioniert allerdings nur, wenn man auch direkt an eine beliebige Stelle des Massenspeichers zugreifen kann, was z.B. bei der Verwendung eines normalen Tonbandes nicht möglich ist. Bei Disketten oder der Harddisk ist dies möglich, da diese in einzelne Abschnitte (Sektoren) unterteilt sind, die durchgehend nummeriert sind.

Zurück zu unserem Beispiel der Adreßverwaltung. Wenn wir wissen, auf welchem Sektor der nullte Datensatz beginnt, können wir auch ausrechnen, wo sich das 640. Byte der Datei befindet. Nehmen wir als Beispiel an, unsere Datei beginnt in Sektor Nummer 10. Beim ATARI ST enthält jeder Sektor 512 Bytes, also werden wir unseren 10. Datensatz bzw. das 640. Byte wohl auf dem 11. Sektor finden, und zwar an der 640-512 = 128.Stelle.

Diese Rechnerei wird dem Programmierer allerdings erspart, wenn er in einer Hochsprache arbeitet. Eine Hochsprache ist eigentlich jede Programmiersprache außer der Maschinensprache, auch Assembler genannt. Assemblerprogrammierer können jedoch auch diese Berechnungen von Sektoren dem Betriebssystem des ATARI ST auftragen, da dieses solche Funktionen zur Verfügung stellt. Doch dazu mehr in Kapitel 7.

Auf dieses einfache Prinzip des Direktzugriffes aufbauend existieren noch einige Variationen von Dateiorganisations-Techniken. So kann man z.B. die ganze Datei nach einem wichtigen Datenfeld, z.B. dem Namen, sortieren, die sortierten Namen in eine eigene Datei mit der jeweiligen Nummer des entsprechenden Datensatzes schreiben. Eine solche Hilfs-Datei nennt man Index-Datei. So entsteht eine aus zwei Dateien bestehende index-sequentielle Datei (Index-Datei mit sequentieller Datei), für die es einige recht raffinierte Suchverfahren gibt.

Dadurch kann man sehr schnell einen bestimmten Datensatz finden und auf ihn zugreifen.

## 2.1 Filestrukturen und Zugriff verschiedener Hochsprachen

Das Betriebssystem eines jeden Rechners stellt die Grundoperationen zum Umgang mit Dateien (Files) zur Verfügung, auf die dann die verschiedenen Hochsprachen ihre Dateiformen aufbauen. Wie erwähnt unterstützt das Betriebssystem des ATARI ST, TOS oder GEMDOS genannt, den Umgang mit RANDOM-ACCESS-Dateien. Diese Dateifunktionen des GEMDOS sollen nun kurz vorgestellt und im Anschluß die Umsetztung in die einzelnen Hochsprachen erläutert werden. Die Beispielprogramme für die verschiedenen Hochsprachen, BASIC, PASCAL, C und FORTRAN, machen alle exakt das gleiche: Anlegen und Lesen einer sequentiellen sowie einer RANDOM-ACCESS-Datei.

### 2.1.1 Die Funktionen des GEMDOS im Überblick

Jede Datei wird vom Anwender durch einen Namen gekennzeichnet. Die maximale Länge des Dateinamens kann 11 Zeichen betragen, wovon die ersten acht vor dem als Trennzeichen fungierenden Punkt den eigentlichen Namen und die letzten 3 eine Zusatzinformation zu dieser Datei (Extender) repräsentieren. Bei der Anwendung von Hochsprache-Compilern, Programmen, die den geschriebenen Programmtext in ein lauffähiges Programm übersetzen, ist die Verwendung von Extendern notwendig. Auf dem Weg vom Programmtext zum fertigen Programm entstehen nämlich bis zu vier verschiedene Files mit gleichem Namen, aber zur Unterscheidung verschiedenen Extendern. Man schreibt z.B. ein C-Programm mit einem Textverarbeitungs-Programm und bennennt das Textfile 'test1.c'. Beim Übersetzen des Programmes entstehen dann Dateien mit den Namen 'test1.o' und 'test1.prg'.

Zum Anlegen einer neuen Datei stellt das GEMDOS die Funktion CREATE (Funktions-Nummer $3C) zur Verfügung. Man übergibt dieser Funktion den gewünschten Dateinamen und ein

sogenanntes Modus-Wort, welches Informationen über die Art der anzulegenden Datei enthält. Hat das Anlegen der Datei geklappt (Diskette nicht schreibgeschützt o.ä.), erhält man vom GEMDOS eine Dateinummer zurück, über die nun alle folgenden Dateizugriffe ablaufen. Diese Nummer nennt man handle.

Die CREATE-Funktion muß nur beim allerersten Zugriff auf eine Datei aufgerufen werden, spätere Zugriffe auf eine schon bestehende Datei können durch Aufruf der Funktion OPEN ($3D) vorbereitet werden. Beim Aufruf von CREATE wird also auf dem aktuellen Laufwerk eine leere Datei mit dem übergebenen Dateinamen angelegt, die anschließend beschrieben werden kann. Viele Hochsprachen übernehmen die CREATE-Funktion in ihre OPEN-Funktion, Beim Aufruf von OPEN, wenn in die Datei geschrieben werden soll, wird dann die Datei neu angelegt, falls noch keine mit dem angegebenen Namen existiert.

Zum Schreiben in eine Datei übergibt man der GEMDOS-Funktion WRITE ($40) die bei CREATE oder OPEN erhaltene Dateinummer (handle), die Anzahl der zu schreibenden Zeichen und natürlich die zu schreibenden Zeichen selbst. Sind alle Daten in die Datei geschrieben und soll in irgendeiner Form wieder auf die Daten zugegriffen werden, muß die Datei vorher geschlossen werden. Die Funktion CLOSE ($3E) übernimmt diese Aufgabe. Wird diese Funktion nicht aufgerufen, gehen meist Daten der Datei verloren, da die Struktur bzw. die Aufteilung der Datei auf der Diskette nicht richtig auf der Diskette vermerkt ist.

Nachdem die Datei durch CREATE angelegt, mittels WRITE beschrieben und schließlich durch CLOSE wieder geschlossen wurde muß diese Datei zum Lesen wieder durch OPEN geöffnet werden. Der OPEN Funktion ($3D) wird ebenso wie bei CREATE der Filename übergeben und zusätzlich noch ein Modus-Wort zwischen 0 und 2.

Eine als Modus übergebene 0 öffnet die Datei nur zum Lesezugriff, d.h. die beim Aufruf erhaltene Dateinummer (handle) ermöglicht nur ein Lesen aus der Datei, versuchte Schreibzugriffe werden mit Fehler abgebrochen. Ein Modus von 1 öffnet die Datei für Nur-Schreib-Zugriff und eine übergebene 2 erlaubt

sowohl Schreib- als auch Lese-Zugriff. Gelesen wird nun mit der Funktion READ ($3F), der analog zu WRITE die Dateinummer aus dem OPEN Aufruf und die Anzahl der zu lesenden Zeichen übergeben werden.

Der Datei-Zugriff mittels READ und WRITE erfolgt rein sequentiell, d.h. beim Öffnen der Datei durch CREATE erzeugt das Betriebssystem einen Zeiger in die Datei, der bei jedem Öffnen und natürlich auch beim Anlegen der Datei wieder auf Null gesetzt wird. Dieser Zeiger weist immer auf die momentan bearbeitete bzw. aktuelle Position innerhalb der Datei, damit man sich leichter zurechtfindet.

Schreibt man nun mittels WRITE z.B. 14 Zeichen in diese Datei, bewegt das Betriebssystem diesen internen Zeiger um 14 Positionen weiter, so daß bei dem nächsten Schreibzugriff die neuen Zeichen an das letzte geschriebene Zeichen angehängt werden. Der Anwender muß also entweder pro Datenfeld eine bestimmte Anzahl von Zeichen zulassen oder ein bestimmtes Zeichen zwischen zwei Datenfelder einfügen, damit beim Lesen das Ende eines Datenfeldes erkannt werden kann.

Für eine Adreßdatei, die ja eine reine Textdatei darstellt, werden nicht alle 256 durch 8-Bit darstellbaren Zeichen (siehe Anhang) benötigt, sondern nur die Klein- und Groß-Buchstaben, die Zahlen sowie die Satzzeichen. Daher existieren im sogenannten ASCII-Zeichensatz (ASCII= American Standard Code of Information Interchange) einige Steuerzeichen, die z.B. das Ende einer Datei oder das Ende eines Datenfeldes markieren.

Genau wie beim Schreiben wird auch beim Lesen von Zeichen aus einer Datei der interne Dateizeiger um die Anzahl der gelesenen Zeichen weiterbewegt. Man kann so zwar jedes Zeichen lesen, muß aber zum Lesen des letzten Zeichens einer Datei alle vorherigen Zeichen durchlesen. Die GEMDOS-Funktion LSEEK ($42) ermöglicht nun die Positionierung des internen Dateizeigers auf ein beliebiges Zeichen relativ zu Dateianfang, Dateiende oder momentanem Dateizeiger. Als Übergabeparameter muß wieder die Dateinummer (handle) sowie ein Modus-Wort

und natürlich die gewünschte Veränderung der Zeigerposition übergeben werden.

Hat dieses Modus-Wort einen Wert von 0, so wird die Position relativ zum Dateianfang berechnet, bei einem Wert von 1 errechnet sich die neue Position des Dateizeigers relativ zum jetzigen Zeiger, d.h. es sind auch negative Werte erlaubt, damit man auch den Zeiger um einige Zeichen zurückbewegen kann. Übergibt man als Modus-Wort eine 2, zählt die Position relativ zum Dateiende, es sind folglich nur negative Werte erlaubt. Mit dieser LSEEK-Funktion wird es möglich, eine RANDOM-ACCESS-Datei mit wahlfreiem Zugriff zu programmieren, indem man feste Datenfeldlängen nimmt, z.B. 13 für den Namen und 64 Zeichen für einen gesamten Datensatz. Dadurch weiß man, um wieviele Zeichen man den internen Dateizeiger bewegen muß, um zum nächtsten oder vorherigen Datenfeld bzw. Datensatz zu gelangen.

Bei der bisherigen Beschreibung der GEMDOS-Funktione fehlen noch drei für Dateihandling wichtige Funktionen:

*SETDTA ($1A)* legt einen Puffer für die beiden Funktionen *SFIRST ($4E)* und *SNEXT ($4F)* fest, mit denen man alle Dateien einer Diskette aus dem Inhaltsverzeichnis lesen und die jeweilige Länge dieser Dateien bestimmen kann.

Nach diesem globalen Einblick in das Betriebssystem des ATARI ST können wir uns nun den einzelnen Hochsprachen zuwenden und die Möglichkeiten des Dateihandlings der jeweiligen Sprachen unter die Lupe nehmen. Diese Beispiele können keine Einführung in die jeweilige Sprache sein und auch keine komplette Dateiverwaltung bieten. Sie sollen nur anhand eines konkreten Beispieles zeigen, wie einach das Anlegen und der Zugriff auf eine Diskettendatei ist. Zum Erlernen der Sprache und auch der Dateiorganisation findet man in der Literatur bereits etliche gute Bücher.

Nach der halb theoretischen Betrachtung der Zugriffstechniken finden Sie dann schließlich im Kapitel 2.6 ein einfaches, aber doch komplettes Datenbank-Programm in BASIC, an dem Sie

die praktische Anwendung der eben erworbenen Kenntnisse sehen können.

## 2.2 Filezugriff von BASIC

Das beim ATARI ST mitgelieferte ATARI-BASIC bietet sowohl den sequentiellen Zugriff auf Dateien als auch den wahlfreien Zugriff. Nach dem Entfernen der Zeilennummern evt. mit dem von GfA mitgelieferten ST-KILL-Programm funktionieren die BASIC-Programme ohne Änderung auch mit dem BASIC-Interpreter von GfA.

### BASIC-Befehlsübersicht

Zum Anlegen einer Diskettendatei verwendet man die Funktion *OPEN*, die drei verschiedene Optionen einer Datei bietet. Die Syntax des Befehls:

OPEN "Modus",#Dateinummer,"Dateinamen",Datensatzlänge

Für *Modus* (unbedingt in Großbuchstaben) existieren folgende Optionen:

"I" Datei soll zum sequentiellen Lesen geöffnet werden.

"O" Datei soll zum sequentiellen Schreiben geöffnet werden.

"R" Datei soll zum wahlfreien Zugriff geöffnet werden.

Die *Dateinummer* ist eine willkürliche Zahl zwischen 1 und 15, ebenso wie der *Dateiname*, der acht Buchstaben gefolgt von einem Punkt und drei weiteren Extension-Buchstaben enthalten kann. Die *Datensatzlänge* spielt nur beim Öffnen einer RANDOM-Datei eine Rolle (Modus = "R") und gibt die Größe jedes Datensatzes in Bytes an. Im Gegensatz zu der Betriebssystemfunktion muß man hier schon beim Öffnen angegeben, ob eine

sequentielle Datei oder eine solche mit wahlfreiem Zugriff angelegt werden soll.

Die Verwendung von sequentiellen Dateien ist in diesem BASIC allerdings stark eingeschränkt, da keine Möglichkeit besteht, Daten an eine schon vorhandene Datei anzuhängen. Dies geht nur mit einem recht unschönen Trick: Hat man z.B. eine sequentielle Adreßdatei mit 100 Adressen gespeichert und möchte nun die von Tante Frieda noch hinzufügen, muß man alle 100 Adressen lesen und zwischenspeichern, anschließend die neue Adresse hinzufügen und alle 101 Adressen wieder in die Datei schreiben.

Jedes OPEN "O" löscht eine schon vorhandene Datei mit gleichem Namen und legt somit eine völlig neue, leere Datei auf der Diskette an. Wegen dieser doch sehr eingeschränkten Handhabung und da die maximale Größe einer sequentiellen Datei von der Größe des Hauptspeichers des ATARI abhängig ist (was für kleinere Datenbanken natürlich kein Hindernis ist), möchte ich nur kurz auf sequentielle Dateien im ATARI-BASIC eingehen.

### 2.2.1 Die sequentielle Datei in BASIC

In eine sequentielle Datei können Zeichenketten (ASCII-Strings) oder auch Zahlen geschrieben werden. Das Schreiben von irgendwelchen Sonderzeichen kann Probleme geben, da dann evtl. das Ende eines Datenfeldes nicht mehr zu finden wäre. Eine solche Datei wird z.B. durch folgenden Befehl zum Beschreiben geöffnet:

OPEN "O",#1,"TEST1.DAT"

Das Schreiben in diese neu angelegte Datei mit dem Namen TEST1.DAT übernehmen die beiden Befehle *WRITE#1* und *PRINT#1*, wobei WRITE zwischen die zu schreibenden Daten ein Komma ausgibt, PRINT dagegen die gleichen Formatierzeichen wie bei der Ausgabe auf den Bildschirm (z.B. Leerzeichen nach einem Komma) benutzt.

PRINT# und WRITE# besitzen die gleiche Schreibweise, und zwar:

PRINT#Dateinummer,Daten[,Daten,...]

WRITE#Dateinummer,Daten[,Daten,...]

Die Datei "TEST1.DAT" könnte man durch folgende Befehlsfolge zum Beschreiben öffnen und Testdaten hineinschreiben:

```
10 open "O",#1,"A:TEST1.DAT"
20 a$ = "Harry"
30 b$ = "Hirsch"
40 for i = 1 to 10
50 write#1,a$
60 write#1,b$
70 next i
80 close #1
```

Dieses kleine Programm legt die Datei "TEST1.DAT" auf der Diskette in Laufwerk A an und schreibt 10 mal "Harry" bzw. "Hirsch" in die Diskettendatei.

Die WRITE-Funktion schreibt einen Text (String) in Anführungszeichen und den Zeichen $0D (CR= Carriage Return) und $0A (LF= Line Feed) als Ende einer Ausgabe auf die Diskette. Das Zeichen $1A dient dem BASIC-Interpreter als Ende-der-Datei Zeichen EOF (End of File) und bietet somit dem Programmierer eine Möglichkeit zur Erkennung des Datei-Endes.

Zum Lesen einer sequentiellen Datei existieren im ATARI-BASIC zwei Befehle, die sich nur hinsichtlich der Behandlung von Steuerzeichen im zu lesenden Text unterscheiden:

Die *INPUT#*-Funktion überliest vorangestellte Leerzeichen CRs, LFs und Sonderzeichen und liest ab dem ersten ASCII-Zeichen bis zu einem Leerzeichen, dem Ende-der-Zeile Zeichen (EOL = End of Line, besteht aus $0A und $0D, LF und CR), einem Komma, dem EOF-Zeichen oder maximal 255 Zeichen. Die *LINE INPUT#*-Funktion liest alle Zeichen vom ersten bis zum

Ende-der-Zeile Zeichen oder bis zu 254 Zeichen. Beiden Befehlen muß eine Variable, in die gelesen werden soll, sowie die Dateinummer der Datei übergeben werden. *INPUT #1,a$* liest einen String aus der Datei mit der Nummer 1 in die Variable a$.

Das folgende kurze Programmfragment öffnet die eben angelegte Datei "TEST1.DAT" zum Lesen und liest alle Strings bis zum Ende der Datei Zeichen (EOF = $1A). Die Funktion *EOF(Dateinummer)* dient zur Erkennung dieses Endes. Sie liefert einen Wahrheitswert: *TRUE* (wahr), wenn das Dateiende erreicht wurde und *FALSE* (unwahr), falls dies nicht der Fall ist.

```
10 open "I",#1,"A:TEST1.DAT"
20 if eof(1) goto 100
30 input #1,a$
40 print a$
50 goto 20
100 close #1
```

### 2.2.2 Die RANDOM-Datei in BASIC

Die Handhabung von RANDOM-Dateien mit wahlfreiem Zugriff ist im ATARI-BASIC wesentlich besser implementiert als der sequentielle Zugriff. Allerdings müssen Sie erst einmal mehrere Befehle kennenlernen, da das Anlegen und Bearbeiten einer RANDOM-Datei auch wesentlich komplexer ist.

Das Öffnen und Anlegen der Datei geschieht noch ohne wesentlichen Unterschied. OPEN #1,"R","TEST2.DAT",64 öffnet die Datei "TEST2.DAT" als Datei mit wahlfreiem Zugriff und vereinbart für diese Datei eine Datensatzlänge von 64 Zeichen bzw. Bytes. Wenn Sie nachher mit *GET#* und *PUT#* auf die Datei zugreifen, geschehen diese Zugriffe immer in "Portionen" zu 64 Zeichen.

Die einzig erlaubten Zeichen dieser Dateiart sind ASCII-Zeichen. Deshalb müssen alle Zahlenwerte, die in eine RANDOM-Datei geschrieben werden sollen, vor dem Schreiben in Zahl-

Zeichen (Ziffern) umgewandelt werden. Beim Lesen muß man diese Zahl-Zeichen wieder in Zahlen zurückwandeln. Aber keine Sorge, für diesen Zweck existieren mehrere BASIC-Funktionen.

Meistens sollen ja in einem Datensatz einer RANDOM-ACCESS-Datei mehrere Datenfelder angelegt werden, z.B. eines für den Namen, eines für den Vornamen ect. (s.o.). Diese Einteilung des vorhandenen Platzes, in diesem Fall der 64 Zeichen, übernimmt die Funktion *FIELD #*. Die Anweisung

```
FIELD #1, 13 AS a$, 13 AS b$, 4 AS c$, 15 AS d$,15 AS e$, 4 AS f$
```

reserviert im 64 Zeichen großen Datensatz-Feld der Datei Nummer eins (#1) 13 Zeichen für a$ (Vorname), 13 für b$ (Name), 4 Zeichen für c$ (Postleitzahl), 15 Zeichen für d$ (Ortsname), 15 Zeichen für e$ (Straßennamen) und 4 Zeichen für f$ (Hausnummer).

Auf diese Stringvariablen sollte nicht direkt, sondern nur über die Funktionen *LSET* und *RSET* zugegriffen werden. LSET a$ = "Harry" überträgt den String "Harry" linksbündig in die Stringvariable a$, die laut obiger Definition 13 Zeichen aufnehmen kann. Die restlichen Zeichen bis zum 13. werden durch Spaces ($20) belegt.

Der Befehl rset a$ = "Harry" füllt die Puffervariable rechtsbündig, d.h der freie Platz pro Datenfeld wird mit führenden Spaces aufgefüllt.

Möchte man Zahlen in eine Random-Datei schreiben, müssen diese zuerst in Byte-Strings umgewandelt werden. Die Funktionen *MKD$*, *MKI$* und *MKS$* übernehmen diese Aufgabe:

mki$(Zahl) gibt einen 2-Byte String aus (für Integer-Variablen)

mks$(Zahl) gibt einen 4-Byte String aus (für Real-Variablen)

mkd$(Zahl) gibt einen 8-Byte String aus (für doppelt genaue Reals)

Zahlen werden also vor der Übertragung in die gewünschte Puffervariable der RANDOM-ACCESS-Datei durch eine dieser Funktionen in einen ASCII-String umgewandelt und später beim Lesen durch eine der Rückverwandlungs-Funktionen (cvi,cvs,cvd) wieder in normal verarbeitbare Zahlen (Reals und Integers) umgeformt.

Nachdem nun die gewünschten Puffervariablen des Datensatzes mittels FIELD angelegt wurden, Strings mit LSET, Zahlen nach vorheriger Anwendung von MKD$ oder MKS$ ect. auch mit LSET in diese Puffervariablen eingetragen wurden, kann der gesamte Datensatz (Name, Vorname ...) mit einem einzigen Befehl in die Datei eingetragen werden: mit PUT. PUT #5, 1 trägt den in den Puffervariablen der Datei Nummer 5 enthaltenen Daten als Datensatz Nummer 1 in die Datei Nummer 5 ein.

Das folgende kleine BASIC-Programm legt eine Random-Datei mit Namen "TEST3.DAT" auf der in Laufwerk A eingelegten Diskette an, spezifiziert 6 Datenfelder für die Puffervariable, belegt diese Puffervariable mit Werten und schreibt diese Werte schließlich als Datensatz Nummer 1 und 2 in die Random-Datei.

```
10 open "R",#1,"A:DATEI3.DAT",64
20 field #1,13 as a$,13 as b$,4 as c$,15 as d$,15 as e$,4 as f$
30 lset a$= "Harry"
40 lset b$= "Hirsch"
50 a = 2222
60 lset c$= mks$(a)
70 lset d$= "Buxtehude"
80 lset e$= "Meerweg"
90 b = 245
100 lset f$=mks$(b)
110 put #1, 1
120 put #1, 2
130 close #1
```

In Zeile 60 wird die Zahl 2222 vor der Zuweisung an die Puffervariable c$ durch MKS$ in einen 4-Byte-String umgewandelt.

Möchte man die Daten der Random-Datei schließlich wieder lesen, geht man analog zum Schreiben vor, indem man die Datei öffnet, die Puffervariablen definiert und einen kompletten Datensatz mit dem Befehl GET#1 einliest. Auf die einzelnen Datenfelder kann dann direkt über die entsprechende Puffervariable zugegriffen werden.

Bei Zahlen muß man an die Rückwandlung denken, da Zahlen ja als ASCII-Strings gespeichert sind. Das folgende kleine BASIC-Programm öffnet die eben angelegte Datei und liest alle Datensätze dieser Datei, welche dann auf dem Bildschirm ausgegeben werden.

```
10 open "R",#1,"A:DATEI3.DAT",64
20 field #1,13 as a$,13 as b$,4 as c$,15 as d$,15 as e$,4 as f$
30 get #1,1
40 print a$,b$
50 print cvs(c$),d$
60 print e$,cvs(f$)
70 close 1
```

Die Feldgrößen der Puffervariablen dürfen beim Schreiben und anschließendem Lesen natürlich nicht voneinander abweichen, d.h. wenn für a$ vor dem Schreiben 13 Zeichen in der Puffervariablen reserviert wurden, müssen beim späteren Lesen auch 13 Zeichen für die Puffervarable a$ reserviert werden.

## 2.3. Das Filehandling in PASCAL

Die Beschreibung der Dateifunktion für PASCAL bezieht sich auf den PASCAL Compiler ST-PASCAL+ von CCD, der direkt von Atari vertrieben wird und eine sehr gute, weit über den PASCAL-Standard hinausgehende, Implementation der Sprache PASCAL auf dem Atari ST darstellt. ST-PASCAL+ unterstützt sowohl sequentielle wie auch RANDOM-ACCESS-Dateien.

Für Dateien verwendet man den Datentyp 'file of' oder den schon vordefinierten Type TEXT, der allerdings nur für sequentielle Dateien angewendet werden kann und dem Typ 'packed array of char' entspricht. Als Beispiel deklariert

var dat: file of integer

eine Datei, die Integer-Zahlen aufnehmen kann und den dazugehörigen Zeiger als Variable 'dat', der auf das aktuell zugegriffene Zeichen innerhalb der Datei zeigt.

### 2.3.1 Die sequentielle Datei in PASCAL

Nach der Deklaration einer Dateivariablen vom Typ 'file of' wird eine neue Datei durch die Funktion rewrite (interner Dateiname, 'externer Name'), die dem BASIC-Befehl OPEN "O" entspricht, angelegt. Durch diesen Befehl wird eine Datei mit dem übergebenen Dateinamen angelegt und einem externen Namen zugeordnet. Der Dateiname muß im Deklarationsteil als Variable des Typs 'file of' deklariert werden. Über den Dateinamen bzw. die ebenfalls durch rewrite definierte Puffervariable (gleicher Name mit angehängtem Hochpfeil ^) kann nun auf die Datei zugegriffen werden.

Der interne Dateiname repräsentiert die Datei innerhalb des PASCAL-Programms und der in Hochkommata stehende externe Name repräsentiert die gleiche Datei auf einem Massenspeicher (Diskettendatei). Deklariert man z.B. die Datei 'dat' durch:

var dat: file of integer ;

und öffnet sie anschließend zum sequentiellen Beschreiben mit rewrite (dat, 'a:sdatei.dat'), so wird gleichzeitig eine Puffervariable dat^ definiert, die eine Integerzahl aufnehmen kann und auf das erste Dateielement zeigt. Außerdem wird auf der Diskettenstation A eine Datei mit Namen "sdatei.dat" angelegt und zum Beschreiben geöffnet. Alle folgenden Aus- und Eingaben beziehen sich dann auf diese Diskettendatei.

Zum Lesezugriff auf eine schon bestehende Datei muß diese mit reset (interner Dateiname, 'externer Name') : wieder geöffnet werden. Durch diesen Befehl wird eine schon existierende Datei zum Lesen geöffnet und gleichzeitig der erste Datensatz in die Puffervariable übertragen. Sollte versucht werden, eine noch nicht bestehende Datei zu öffnen, wird EOF() wahr.

Die Funktion EOF (interner Dateiname) : gibt einen Wahrheitswert vom Typ Boolean (TRUE, FALSE) zurück. TRUE wird zurückgegeben wenn der Dateizeiger auf das Ende der Datei zeigt.

EOL (Dateivariable) : ist ebenfalls eine Funktion vom Typ Boolean, die jedoch nur auf Dateien des Typs 'packed file of char' bzw. TEXT angewendet werden darf und TRUE zurückgibt, wenn das Ende einer Zeile erreicht wurde.

Der Zugriff auf die Daten einer Datei erfolgt schließlich über put (interner Dateiname) : zum Schreibzugriff und get (interner Dateiname) zum Lesezugriff.

put (dat) schreibt den Wert der Puffervariable dat^ in die Datei. Die Puffervariable stellt praktisch einen Zeiger in die Datei dar, der bei jedem rewrite oder reset auf Null gesetzt und bei jedem Zugriff mit get oder put um eins erhöht wird und somit auf das nächste Element der Datei zeigt. Nach dem Öffnen der Datei zum Lesen durch reset (dat,'name') wird schon das erste Dateielement in die Puffervariable dat^ übertragen. Ein nachfolgendes get (dat) erhöht den Dateizeiger um eins und überträgt den Wert, auf den der Zeiger dann zeigt, in die Puffervariable dat^. Zum Erkennen des Dateiendes dient die Funktion eof (Dateivariable), die einen +Wert vom Typ Boolean (TRUE,FALSE) liefert.

Im Beispiel muß vor dem Zugriff mit get auf Dateiende getestet werden, da get den Dateizeiger erhöht und das nächste Dateielement in die Puffervarable übertragen will. Bei Dateien vom Typ TEXT besteht zusätzlich die Möglichkeit, das Zeilenende

durch die Funktion eol (Dateivariable) zu erkennen, die ebenso wie EOF einen Wert vom Typ Boolean zurückgibt.

Als mögliche Dateielemente können sämtliche in PASCAL verfügbaren Datentypen dienen, selbstverständlich auch RECORDS. Nachdem eine Datei also mit rewrite geöffnet wurde, kann der Puffervariablen ein Wert zugeordnet werden, der anschließend durch put in die Datei geschrieben wird. Für reine Textdateien, also solche vom Typ packed 'file of char' (TEXT), kann die zum Schreiben eines Dateielementes eigentlich nötige Befehlsfolge (Zuweisung eines Wertes an die Puffervariable) durch dat^ := wert; und Schreiben dieses Wertes in die Datei durch put(dat); durch den einzelnen Befehl write (dat, wert); abgekürzt werden. Analog hierzu liest der Befehl read (dat, wert) aus einer TEXT-Datei und ersetzt die Befehle wert := dat^ und get (dat).

Das nun folgende kleine PASCAL-Programm legt eine Datei auf der Diskette in Laufwerk A an und beschreibt diese mit 20 Strings. In CCD-PASCAL definiert string[20] eine Variable vom Typ 'packed array of char', die 21 Zeichen aufnehmen kann. Der PASCAL-Compiler merkt sich die Länge jedes Strings, indem er diesen Wert an den Anfang des Strings, also ins nullte Zeichen einsetzt.

```
(* Schreiben einer sequentiellen Datei in PASCAL. U.B. 9.86 *)

program sdatei ;

var   dat1 : file of string[20] ;
      t1,t2 : string[20] ;
          i : integer ;

begin
    rewrite (dat1, 'a:seqdatei.dat' );
    t1 := 'Harry';
    t2 := 'Hirsch';

    for i:= 1 to 10 do
      begin
```

```
            dat1^ := t1;
            put (dat1);
            dat1^ := t2;
            put (dat1);
        end; (* for Schleife *)

end. (* Programm *)
```

Wenn Sie sich mit dem in Kapitel 7 vorgestellten Disk-Monitor die angelegte Datei "seqdatei.dat" ansehen, so erkennen Sie deutlich das Organisationsschema einer sequentiellen PASCAL-Datei mit Stringvariablen (21 Zeichen pro String reserviert, Länge des Strings am Anfang des Strings). Zum Lesen der soeben angelegten Datei dient folgendes kleine Programm:

```
(* Lesen einer sequentiellen Datei in PASCAL. U.B. 9.86 *)


program liesdatei ;

var   dat1 : file of string[20] ;
     t1,t2 : string[20] ;
         i : integer ;


begin
   writeln (' Datei lesen ');
   reset (dat1,'a:seqdatei.dat');

   while not eof(dat1) do
     begin
        t1 := dat1^;
        get (dat1);
        writeln (t1);
     end; (* while Schleife *)
   writeln;
```

```
        writeln (' Return- Taste betätigen ');
        readln (t2);

    end. (* Programm *)
```

Nach dem Öffnen der Datei mit reset(dat1,'A:psequt1.dat') wird das erste Dateielement schon der Puffervariablen dat1^ zugeordnet, so daß die Puffervariable schon direkt nach dem Öffnen der Datei einer Variablen verarbeitet werden kann. Diese Variable muß natürlich vom gleichen Typ wie die durch die Deklaration der Dateivariable mitdefinierte Puffervariable sein, sonst können Fehler auftreten. Außerdem darf kein Versuch unternommen werden, Daten hinter dem Dateiende zu lesen, die Funktion eof (dat1) fragt ab, ob das Dateiende schon erreicht ist; die Leseschleife wird in diesem Fall verlassen.

In PASCAL existiert ebenso wie in BASIC keine Möglichkeit, Daten ans Ende einer bestehenden sequentiellen Datei anzuhängen. Möchten Sie eine schon existierende Datei erweitern, bleibt Ihnen nichts anderes übrig, als die gesamte Datei zu lesen und mit dem neuen Dateielement in eine neu anzulegende Datei zu schreiben.

Das Anlegen und der Zugriff auf Dateien anderer Datentypen (file of integer, file of real) geschieht analog zu obigem Beispiel.

### 2.3.2 Random-Dateien in PASCAL

Das Anlegen und Öffnen zum Lesen von RANDOM-ACCESS Dateien geschieht mit den gleichen Befehlen wie bei den sequentiellen Dateien (rewrite, reset), auch der Zugriff auf die einzelnen Daten ist ähnlich. Es gibt nur einen zusätzlichen Parameter bei get und put, nämlich die Nummer des Datensatzes, der geschrieben oder gelesen werden soll. Die Nummerierung der Datensätze beginnt bei 0, wobei alle Datensätze zwischen 0 und der größten Nummer erst angelegt werden müssen.

Wenn also z.B. der letzte Datensatz die Nummer 8 trägt, kann danach kein Datensatz mit der Nummer 10 angelegt werden,

sondern es muß zuerst der Datensatz Nummer 9 geschrieben werden. An einem kleinen Beispielprogramm kann man die Flexibilität dieses Dateityps deutlich erkennen. Es wird hier eine kleine Adreßdatei angelegt, in die 10 mal die gleiche Adresse eingetragen wird.

```
(* Schreiben einer Random-Datei in PASCAL. U.B. 9.86 *)


program randatei ;

type adres =
   record
      vorname : string[12];
         name : string[12];
          plz : integer;
          ort : string[14];
      strasse : string[14];
       nummer : integer;
   end; (* record *)

var   dat1 : file of adres ;
     t1,t2 : adres ;
         i : integer ;

begin
   rewrite (dat1,'a:random1.dat');
   t1.vorname := 'Harry';
   t1.name := 'Hirsch';
   t1.plz := 2222 ;
   t1.ort := 'Buxtehude';
   t1.strasse := 'Meerweg';
   t1.nummer := 245;

   for i:= 0 to 9 do
     begin
        dat1^ := t1;
```

```
            put (dat1,i);
        end; (* for Schleife *)

    end. (* Programm *)
```

Mit dem Befehl dat1^ := t1 wird also im CCD-PASCAL die gesamte Adreßstruktur mit Vorname, Name ect an die Puffervariable übergeben und anschließend mit put (dat1,i) als jeweiliger Datensatz Nummer i in die Datei geschrieben.

Wie Sie erkennen können, werden wieder die Anzahl der Zeichen der einzelnen Strings vor dem ersten Zeichen des jeweiligen Strings gespeichert, einfache Ingegerzahlen werden als 2-Byte Sedezimalzahlen gespeichert. Als Ende-der-Datei-Zeichen verwendet PASCAL schließlich die Zahl $F5.

## 2.4 Der Dateizugriff von C

Die Sprache C ist sozusagen die Muttersprache des Atari ST. Große Teile seines Betriebssystems sind in dieser Sprache geschrieben worden. So ist es nicht verwunderlich, die in der Einführung zu diesem Kapitel beschriebenen GEMDOS-Funktionen zur Dateiverwaltung in der Sprachbeschreibung von C, teils in abgewandelter Form, wiederzufinden.

C ist die aus Anwendersicht unvollkommenste Sprache, da viele Funktionen selbst gebaut werden müssen. So auch die Funktionen für die Dateiverwaltung. Alle C-Compiler Anbieter liefern jedoch den im C-Standardbuch von Kernighan & Ritchie beschriebenen Dateistandard als Include-Datei "STDIO.H" mit.

Zur Benutzung der Dateifunktionen muß diese Datei daher durch den am Anfang eines C-Programmes stehenden Befehl #include <stdio.h> ins eigene Programm integriert werden.

Als Hürde für einen C-Anfänger, neben den total chaotisch ausschauenden Kurzzeichen (&,!=,~,||), ist beim Atari ST der C-Compiler von Digital-Research anzusehen, jedenfalls die ersten

Versionen. Ein unerfahrener und somit unsicherer C-Anfänger fragt sich immer wieder "Ist das jetzt mein eigener Fehler oder ein Fehler im Compiler?", wenn wieder mal ein selbstgeschriebenes Programm während oder auch nach dem Compilieren den Rechner zum Absturz bringt. Aus diesem Grunde sind alle hier vorgestellten kleinen C-Programme mit dem LATTICE-C-Compiler von Metacomco compiliert worden.

Die Anpassung an andere C-Compiler dürfte keine Schwierigkeiten bereiten, da nur die Standard-Funktionen der STDIO.H Bibliothek verwendet wurden (Programme funktionieren ohne Änderung mit Megamax).

Die Kommunikation mit Dateien geht in C über eine Datenstruktur vom Typ FILE, die, wie auch die Zugriffsfunktionen auf diese Datenstruktur, in der STDIO.H Bibliothek definiert ist. Hier sind ebenfalls die verschiedene Parameter der Datei wie z.B. die Adresse des Dateipuffers oder den momentanen Zeiger in diesen Puffer enthalten. Hier nun ein Überblick über die einzelnen Zugriffsfunktionen mit dem Datentyp ihrer Parameter:

*pointer = fopen (name, modus)*

```
FILE    *fopen()
FILE    *pointer
char    *name
char    *modus
```

Die möglichen Modusworte :

"w" : anlegen einer Datei und öffnen zum Schreiben
"a" : öffnen einer vorhandenen Datei zum Anfügen von Daten
"r" : Öffnen einer vorhandenen Datei zum Lesen von Daten

Außer diesen existieren noch andere Modus-Worte, die je nach Compiler verschiedene Funktionen erfüllen und für unsere Zwecke nicht weiter wichtig sind.

Die Funktion öffnet eine Datei zum nachfolgenden Zugriff je nach Moduswort. Bei einem auftretenden Fehler ist pointer = NULL, andernfalls wird in pointer der Zeiger auf die Datei zurückgegeben.

*code = fclose (pointer)*

int code
FILE *pointer

Schließt die Datei, auf die pointer zeigt.

*fprintf (pointer,format,argumente)*

FILE *pointer
char *format
char *argumente

Schreibt beliebig viele, durch Kommata abgetrennte Argumente (Strings) mit dem durch den'format' Parameter beschriebenen Format in eine Datei. Die Format-Parameter entsprechen denen des normalen printf-Befehls.

*code = fscanf (pointer, format, chpointer)*

FILE *pointer
char *format
char *chpointer
int code

Liest Zeichenketten aus der durch pointer spezifizierten Datei in dem durch format angegebenem Format in die Variable chpointer. Die Format-Optionen sind identisch mit denen des scanf-Befehls.

*code = fputs (buffer,pointer)*

FILE *pointer
char *buffer
int code

Schreibt den Character-String, auf den buffer zeigt, bis zum Nullbyte in die Datei, auf die pointer zeigt. Bei einem Fehler ist code = EOF. Das Nullbyte, mit dem jeder C-String endet, wird nicht mitgeschrieben, sondern ein NEWLINE-Zeichen.

*code = fgets (buffer, anzahl, pointer)*

FILE *pointer
char *chpoint
char *buffer
int anzahl
int code

Liest anzahl Zeichen aus der Datei, auf die pointer zeigt, in den Puffer, auf den buffer zeigt. Das Lesen wird auch bei einem auftretenden End of Line Zeichen (EOL) beendet. An die eingelesene Zeichenkette wird eine Null-Byte angehängt und der Zeiger auf den Puffer in chpoint übergeben. Nach einem fehlerfreien Zugriff zeigt demzufolge chpoint auf buffer, andernfalls enthält chpoint eine 0, was in C durch NULL ausgedrückt wird.

*code = fputc (chpoint, pointer)*

FILE *pointer
char *chpoint
int code

Schreibt ein einzelnes Zeichen, auf das chpoint zeigt, in die Datei, auf die pointer zeigt. Nach einem Fehler ist code = EOF, sonst der Code des geschriebenen Zeichens.

*code = fgetc (pointer)*

FILE *pointer
int code

Liest ein einzelnes Zeichen aus der Datei, auf die pointer zeigt. Der Code des gelesenen Zeichens wird in code übergeben oder EOF, wenn das Dateiende erreicht wurde.

*code = fseek (pointer,position,mode)*

FILE *pointer
long position
int mode
int code

Stellt den Dateizeiger der Datei, auf die pointer zeigt, auf einen neuen Wert ein. Der Mode-Parameter spezifiziert die neue Position des Zeigers und kann folgende Werte annehmen:

0 : neue Position relativ zum Dateianfang einstellen.
1 : neue Position relativ zur jetzigen Position einstellen.
2 : neue Position relativ zum Dateiende einstellen.

### 2.4.1 Die sequentielle Datei in C

Das nachfolgende C-Programm öffnet die Datei 'SEQDATEI.DAT' zum Schreiben und schreibt 10 mal "Harry Hirsch" in diese Datei.

```
/* Schreiben einer sequentiellen Datei in C. U.B. 9.86 */

#include <math.h>
#include <stdio.h>
```

```
main ()
{
   int i, k ;
   FILE *dat1, *fopen() ;

   char *t1 = "Harry" ;
   char *t2 = "Hirsch" ;

   dat1 = fopen("a:seqdatei.dat","w") ;

   for ( k=1; k < 11; k++)
   {
      fprintf (dat1,"%13s",t1);
      fprintf (dat1,"%13s",t2);

   } /* Ende for-Schleife */

   i = fclose (dat1);
   printf (" Taste betaetigen\n ");

   getchar();
} /* Ende main */
```

Zum Lesen der eben geschriebenen Datei dient das folgende Programm, das den Inhalt der gesamten Datei auf den Bildschirm schreibt.

```
/* Lesen einer sequentiellen Datei in C. U.B. 9.86 */

#include <stdio.h>

main ()
{
   int i, k ;
   FILE *dat1, *fopen() ;

   char platz1 [14] ;
   char *p ;
```

```
   dat1 = fopen("a:seqdatei.dat","r") ;

   while (p = fgets (platz1, 14, dat1) != NULL)

   {
      printf ("%s\n",platz1 );

   } /* Ende while-Schleife */

   i = fclose (dat1);
   printf("\n\n");

   printf (" Taste betaetigen\n ");

   getchar();
} /* Ende main */
```

### 2.4.2 Die Random-Datei in C

Um den wahlfreien Zugriff auf eine C-Datei zu ermöglichen, benötigt man die Funktion fseek(), die das Positionieren des Dateizeigers auf ein bestimmtes Zeichen innerhalb der Datei ermöglicht. Durch das formatierte Schreiben in die Datei mit fprintf() erhält jedes Datenfeld eine festgelegte Länge z.B. 13 Zeichen für den Vornamen usw. Somit hat natürlich auch jeder komplette Datensatz, z.B. eine Adresse, eine genau festgelegte Länge (in dieser Datei 64 Zeichen). Zum Lesen des zehnten Datensatzes muß man nur die Länge eines Datensatzes mit der Nummer des gewünschten Datensatzes multiplizieren, den Dateizeiger auf den errechneten Wert einstellen und kann dann den gewünschten Datensatz bearbeiten. In C beginnt die Nummerierung der Datensätze mit null.

```
/* Schreiben einer RANDOM-Datei in C. U.B. 9.86 */


#include <math.h>
#include <stdio.h>
```

```
char *vorname = "Harry";
char *name = "Hirsch";
char *ort = "Buxtehude" ;
char *strasse = "Meerweg";
int plz = 2222 ;
int nummer = 264 ;


main ()
{
   int i, k ;
   FILE *dat1, *fopen() ;

   dat1 = fopen("A:random2.dat","w") ;

   for ( k=1; k < 11; k++)
   {
      fprintf (dat1,"%13s",vorname);
      fprintf (dat1,"%13s",name);
      fprintf (dat1,"%4d",plz);
      fprintf (dat1,"%15s",ort);
      fprintf (dat1,"%15s",strasse);
      fprintf (dat1,"%4d",nummer);

   } /* Ende for-Schleife */

   i = fclose (dat1);
   printf (" Taste betaetigen\n ");

   getchar();
}  /* Ende main */
```

Das folgende Programm liest alle Daten der Datei und zeigt sie auf dem Bildschirm an, inclusive Datensatznummer und relative Position innerhalb der Datei.

```
/* Lesen einer Random-Datei in C. U.B. 9.86 */


#include <math.h>
#include <stdio.h>

#define LAENGE 64L


main ()
{
   int k, i1, i ;
   FILE *dat1, *fopen() ;
   long pos ;

   char platz1[80], *p ;

   dat1 = fopen("a:random2.dat","r") ;

   k = 0;
   pos = k*LAENGE;

   while ((i = fgetc(dat1)) != EOF)
   {
      i = fseek(dat1,pos,0);

      printf(" Datensatznummer = %8d\n",k);
      printf(" Bytepos. in Datei = %8d\n",pos);
      printf("\n");

      p = fgets(platz1,14,dat1);
      printf(" Vorname = %s\n",platz1);

      p = fgets(platz1,14,dat1);
      printf(" Name = %s\n",platz1);

      p = fgets(platz1,5,dat1);
      i1 = atoi(platz1);
      printf(" Postleitzahl = %8d\n",i1);
```

```
            p = fgets(platz1,16,dat1);
            printf(" Wohnort = %s\n",platz1);

            p = fgets(platz1,16,dat1);
            printf(" Strasse = %s\n",platz1);

            p = fgets(platz1,5,dat1);
            i1 = atoi(platz1);
            printf(" Hausnummer = %8d\n",i1);

            k+=1;
            pos=k*LAENGE;

            printf("**************************\n\n");

         } /* Ende WHILE-Schleife */

         i = fclose (dat1);
         printf("\n\n");

         printf (" Taste betaetigen\n ");

         getchar();

      } /* Ende main */
```

## 2.5 Das Filehandling in FORTRAN

Alle Ausführungen über die Sprache FORTRAN beziehen sich auf den PRO FORTRAN-77-Compiler von PROSPERO, der über die Firma FOCUS vertrieben wird. Dieses FORTRAN ermöglicht, ebenso wie CCD-PASCAL, das Arbeiten mit sequentiellen und auch mit RANDOM-ACCESS-Dateien. Die Implementation auf den Atari ST kann man als sehr gelungen bezeichnen, da alle Sprachdefinitionen des 77'er FORTRAN Stan-

dards einbezogen wurden. Aber auch in Punkto Rechengeschwindigkeit läßt dieser Compiler, zumindest bei mathematischen Berechnungen, die verschieden C-Compiler und auch den CCD-PASCAL-Compiler hinter sich.

### 2.5.1 Die sequentielle Datei in FORTRAN

Zum Öffnen und auch Anlegen einer sequentiellen Diskettendatei bedient man sich der open-Funktion, die viele meist optionale Parameter hat. open (5, file = 'a:fdat1.dat') öffnet eine Datei mit Zugriffs-kanal Nummer 5 und Namen "fdat1.dat" auf dem Laufwerk A. Sollte diese Datei noch nicht existieren, wird sie neu angelegt.

Zum Schreiben in diese Datei können die normalen Ein-Ausgabe-Befehle read und write mit optionalen Parametern verwendet werden. write (5) 'Harry' schreibt in die Datei Nummer 5. Die Ausgabe mit write unterliegt den üblichen FORTRAN-Format Möglichkeiten, deren eingehende Beschreibungen wohl den Rahmen dieses Buches sprengen würden.

Hier nun unser Beispielprogramm, welches eine sequentielle Datei anlegt und 10 mal den Namen "Harry Hirsch" hineinschreibt, in FORTRAN:

```
program seq1

character*13 name , vornam

vornam = 'Harry'
name = 'Hirsch'

open (2, file = 'a:fseque1.dat',form='unformatted')

do 100 n = 1,10
write (2) vornam
write (2) name
```

```
100 continue
    close (2)
    end
```

Zum Lesen dieser sequentiellen Datei bedient man sich des folgenden Programms:

```
    program seq2
    character*2 t1
    character*13 text

    open (2,file='a:fseque1.dat',form='unformatted',status='old')
100 continue
    read (2,end=200) text
    write (*,*) text
    goto 100

200 continue
    close (2)
    end
```

### 2.5.2 Die RANDOM-Datei in FORTRAN

Und nun wieder unser Standard-Programm für RANDOM-ACCESS-Dateien, diesmal in FORTRAN:

```
C Schreiben einer Random-Datei in FORTRAN. U.B. 9.86


    program rand1
    integer*4 plz, nummer

    character*13 name , vornam
    character*15 strass, ort

    vornam = 'Harry'
```

```
      name = 'Hirsch'
      plz = 2222
      ort = 'Buxtehude'
      strass = 'Meerweg'
      nummer = 264

      open (2, file = 'a:\frand1.dat', recl = 64, access='direct')
      do 100 n = 1,10
      write (2,rec= n) vornam, name, plz, ort, strass, nummer
  100 continue
      close (2)
      end
```

Zum Lesen dieser Daten aus der Datei dient das nächste kleine Programm:

```
C Lesen einer Random-Datei in FORTRAN. U.B. 9.86


      program rand1
      integer*4 plz, nummer, stat

      character*13 name , vornam
      character*15 strass, ort

      open (2, file = 'a:\frand1.dat', recl = 64, access='direct',
     - status = 'old')

      n = 1
   10 continue

      read (2,rec=n, iostat = stat ) vornam, name, plz,
     - ort, strass, nummer
      if (stat .eq. 0) then

      write (*,*) ' Datensatz Nummer: ',n
      write (*,*)
      write (*,*) ' Vorname = ' , vornam
```

```
write (*,*) ' Name = ', name
write (*,'(a,i6)') ' Postlz. = ' , plz
write (*,*) ' Wohnort = ', ort
write (*,*) ' Strasse = ' , strass
write (*,'(a,i6)') ' Hausnr. = ' , nummer
write (*,*)
write (*,*)
n = n+1

goto 10

else
write (*,*)
write (*,*)
write (*,*) ' Taste betaetigen '
close (2)
endif
end
```

## 2.6 Eine einfache Datenbank

Im Anschluß an all diese Theorie wollen wir nun einmal die eben erworbenen Kenntnisse anhand einer kleinen Datenbank erproben. Dieses Programm ist nicht unbedingt für die Lagerverwaltung eines Warenhauses anwendbar, reicht aber für die Verwaltung von Telefon-Nummern oder einer Schallplatten-Sammlung völlig aus.

Das Programm ist in BASIC geschrieben, und zwar für den beim ATARI ST mitgelieferten Interpreter. Dieses BASIC ist bekanntermaßen voller Fehler (Stand 10/86), so daß es empfehlenswert ist, das Programm z.B. für Gfa-BASIC umzuschreiben. Es läuft jedoch in dieser Form auch vollständig in dem ATARI-BASIC.

Bei der Erstellung eines solchen Programmes ist zuerst zu überlegen, was ein Datenbank-Programm alles können soll. Die wichtigsten Funktionen sind in dem vorgestellten Programm enthalten, und zwar:

- Erstellung einer neuen Datenbank

- Eingabe von neuen Daten bzw. Korrektur alter Einträge

- Einladen einer bereits angelegten Datenbank von Diskette

- Ausgabe der Daten auf dem Bildschirm oder dem Drucker

- Suchen nach bestimmten Schlüsselworten

- Sortieren der Daten nach einem beliebigen Datenfeld

- Beendigung des Programmes

Diese Funktionen sind über ein einfaches Menü aufrufbar, welches auf dem Bildschirm dargestellt wird. Hierfür wird nur die Funktions-Ziffer eingegeben mit nachfolgendem Return.

Bevor wir die Bedienung der einzelnen Funktionen näher betrachten, wäre es sinnvoll, erst einmal das folgende Programm abzutippen:

```
10   '*** Mini-Datenbank S.D. ***
20   dim d$(5),i$(5),l(5),p$(500),r(500)
30   for i=1 to 500: r(i)=i: next i
40   for i=1 to 5: d$(i)=space$(100)
50   i$(i)="": next i

60   start:
70   fullw 2: clearw 2: gotoxy 0,0
80   ? "**** Mini-Datenbank aus: Floppy-Buch zum ST S.D. ****
90   ?: ? d;" Datensätze vorhanden in Datei ";f$
100  for i=1 to 5
110  gotoxy 22,1+i: ?i;") ";i$(i)
120  next i
130  if so then gotoxy 21,1+so: ?">"
```

```
140  gotoxy 0,5
150  ?: ? "1) Anlegen einer Datenbank"
160  ? "2) Eingabe von Daten"
170  ? "3) Laden einer Datenbank"
180  ? "4) Sortieren der Daten"
190  ? "5) Suchen"
200  ? "6) Ausgabe von Daten"
210  ? "7) Ende"
220  ?: input "Ihre Wahl ";w
230  on w gosub anlegen,eingabe,laden,sortieren,
     suchen,ausgabe,ende240 goto start

250  '
260  '** Anlegen einer Datenbank **
270  anlegen:
280  ? " ** Datenbank anlegen: 500 Einträge mit 5 Feldern frei **"
290  sum=0
300  ?: for i=1 to 5
310  ? i;". Feldname, Länge ";
320  input i$(i),l(i)
330  sum=sum+l(i)
340  next i
350  ?: input "OK ";o$
360  if o$="n" or o$="N" then anlegen
370  gosub getfn
380  open "O",#1,fi$
390  for i=1 to 5
400  print#1,i$(i)
410  print#1,l(i)
420  d$(i)=space$(l(i))
430  next i
440  close #1
450  open "R",#1,fd$,sum
460  field #1, l(1) as d$(1), l(2) as d$(2), l(3) as d$(3), l(4)
     as d$(4), l(5) as d$(5)470 d=0
480  return

490  '
500  '** Eingabe von Daten **
510  eingabe:
```

```
520  clearw 2: gotoxy 0,0: ? " *** Dateneingabe ***
530  ? d;" Datensätze vorhanden"
540  gotoxy 0,3:? "Nummer ";d+1
550  gotoxy 0,3: input "Nummer ";d$
560  if len(d$)>0 then d1=val(d$) else d1=d+1
570  if d1=0 then return
580  if d1>d+1 then eingabe
590  if d1<d+1 then gotoxy 0,5: o$="b": gosub ausgabe1
600  for i=1 to 5
610  gotoxy 0,4+i
620  ?i$(i);: gotoxy 20,4+i
630  input d$
640  if len(d$)>0 then lset d$(i)=d$
650  next i
660  ?: input "OK (j/n) ";o$
670  if o$="n" or o$="N" then eingabe
680  if d1=d+1 then d=d+1
690  put #1,r(d1)
700  goto eingabe

710  '
720  '** Datenbank laden **
730  laden:
740  gosub getfn
750  close #1
760  sum=0
770  open "I",#1,fi$
780  for i=1 to 5
790  input#1,i$(i)
800  input#1,l(i)
810  sum=sum+l(i)
820  d$(i)=space$(l(i))
830  next i
840  close #1
850  open "R",#1,fd$,sum
860  field #1, l(1) as d$(1), l(2) as d$(2), l(3) as d$(3), l(4)
     as d$(4), l(5) as d$(5)870 d=0
880  while not eof(1)
890  d=d+1
900  get #1,d
```

```
910  wend
920  return

930  '
940  '** Datenausgabe **
950  ausgabe:
960  if d=0 then ? "Keine Daten vorhanden !": goto waitkey
970  ? " ** Datenausgabe **"
980  input "B)ildschirm oder D)rucker ";o$
990  for d1=1 to d
1000 gosub ausgabe1
1010 if o$="d" or o$="D" then lprint else ?
1020 next d1
1030 waitkey:
1040 ?: input "-Bitte 'Return' drücken-",w$
1050 return
1060 ausgabe1:
1070 get #1,r(d1)
1080 for j=1 to 5
1090 if o$="d" or o$="D" then lprint i$(j),d$(j) else ?
     i$(j),d$(j)
1100 next j
1110 return

1120 '
1130 '** Suchen **
1140 suchen:
1150 if d=0 then ? "Keine Daten vorhanden !": goto waitkey
1160 ?: input "Feldnummer, Text ";f,t$
1170 for d1=1 to d
1180 get #1,d1
1190 if instr(d$(f),t$) then gosub ausgabe1: ?
1200 next d1
1210 goto waitkey

1220 '
1230 '** Sortieren **
1240 sortieren:
1250 if d=0 then ? "Keine Daten vorhanden !": goto waitkey
1260 ?: input " Nach welchem Feld sortieren ";so
```

```
1270 if so=0 or so>5 then return
1280 for i=1 to d
1290 get #1,i
1300 p$(i)=d$(so)
1310 next i
1320 for i=1 to d
1330 for j=i to d
1340 if p$(r(i))>p$(r(j)) then swap r(i),r(j)
1350 next j
1360 next i
1370 return

1380 '
1390 '** Ende **
1400 ende:
1410 close #1
1420 ?: ? "**** Ende. Tschüs! ****"
1430 end

1440 '
1450 '** Unterprogramme **
1460 getfn:
1470 ?: input "Dateiname ";f$
1480 fi$=f$+".idx"
1490 fd$=f$+".dat"
1500 return
```

Nun zu den einzelnen Funktionen:

## 1. Anlegen einer Datenbank

Nach Aufruf dieser Funktion wird fünfmal zur Eingabe zweier Daten aufgefordert: Feldname und Länge. Hier wird jeweils der Name des Datenfeldes und, durch Komma getrennt, die maximale Länge dieser Einträge in Zeichen eingegeben. Für eine Adreßverwaltung könnte z.B. eingegeben werden:

Name,15
Vorname,12

Ort,16
Strasse,16
Telefon,10

Hat man diese Daten eingegeben, so wird sicherheitshalber noch einmal gefragt 'OK ?'. Sind die eingegebenen Daten in Ordnung, so geben Sie hier einfach J ein (Groß- oder Kleinschreibung spielt dabei keine Rolle).

Als nächstes wird nach dem Dateinamen gefragt, unter dem die Datenbank auf der Diskette abgespeichert werden soll. Erlaubt ist hier die Eingabe des Laufwerks mit dem Namen, z.B. A:TEST. Ein Extender (z.B. .DAT) darf nicht eingegeben werden, da das Programm zwei Dateien unter dem selben Namen mit unterschiedlichen Extendern anlegt. Sie finden daher nachher auf der Diskette eine Datei mit dem Extender .IDX, in der die soeben eingegebenen Namen und Längen der Datenfelder abgespeichert sind sowie eine mit .DAT, welche die Datensätze selbst enthält.

Sind die Daten eingegeben und die Dateien auf Diskette angelegt, wird wieder das Hauptmenü angezeigt.

2. Eingabe von Daten

Nach Anwahl dieser Funktion erhält man die Information, wie viele Datensätze bisher vorhanden sind, und wird nun aufgefordert, die zu ändernde bzw. einzugebende Datensatznummer einzugeben. Die Nummer des nächsten freien Datensatzes steht bereits direkt hinter dem Fragezeichen, so daß Sie für die Eingabe eines neuen Datensatzes nur Return drücken brauchen.

Wollen Sie einen Datensatz ändern, so geben Sie hier dessen Nummer ein. Sie erhalten nun sowohl den alten Inhalt des Datensatzes sowie ein Fragezeichen, welches zur Eingabe der neuen Daten auffordert. Wollen Sie den alten Inhalt eines Datenfeldes übernehmen, so drücken Sie nur Return.

Die Eingabe aller weiteren Daten läuft genau so ab. Wollen Sie die Eingabe beenden, so brauchen Sie nur bei der Frage nach der Datensatz-Nummer eine 0 einzugeben: Sie finden sich sofort wieder im Hauptmenü wieder.

3. Laden einer Datenbank

Hier wird lediglich zur Eingabe des Namens der Datenbank aufgefordert. Auch hier sind nur die Angabe des Laufwerks und der Name selbst erlaubt. Ist die Datenbank geladen, so wird wieder das Hauptmenü angezeigt, in dem nun auch der Dateiname, die Anzahl der vorhandenen Einträge sowie die Feldnamen mit ihren Nummern aufgelistet werden.

4. Sortieren der Daten

Wollen Sie die Datensätze nach einem bestimmten Feld sortiert ausgeben lassen, so wählen Sie diese Funktion. Es wird nun nach der Feldnummer gefragt, nach der sortiert werden soll. So können Sie z.B. Ihre Adressenliste nach Namen sortieren, ausdrukken und dann noch einmal nach Orten sortieren und ebenfalls ausgeben.

Die Sortierfunktion enthält keine Ausgabefunktion. Wollen Sie wissen, wonach Sie zuletzt sortiert haben, so finden Sie diese Information im Hauptmenü, wo vor dem gewählten Feldnamen ein '>'-Symbol steht.

5. Suchen

Diese Funktion fordert Sie auf, die Feldnummer und den Suchstring einzugeben. Wollen Sie also z.B. alle Adressen aus Buxtehude ausgeben, so geben Sie bei obigem Beispiel der Datenbank 3,Buxtehude ein. Nun werden alle Datensätze, deren Ortsangabe

(Feld Nummer 3) den Text 'Buxtehude' enthält, angezeigt. Sie können hier auch nur Bux als Suchstring eingeben, da es wohl kaum noch andere Ortsnamen mit diesem Anfang gibt.

6. Ausgabe von Daten

Diese Funktion ermöglicht die Ausgabe aller Datensätze auf dem Bildschirm oder dem Drucker. Die Abfrage, wohin ausgegeben werden soll, wird für das Drucken mit D oder d beantwortet, alle anderen Eingaben bewirken die Bildschirmausgabe.

Die Ausgabe erfolgt in der Reihenfolge, in der die Datensätze eingegeben wurden, es sei denn, Sie haben vorher die Sortierfunktion aufgerufen.

7. Ende

Hier wird nur der geöffnete Datenkanal geschlossen (CLOSE #1) und nach einer Abschiedsmeldung das Programm beendet.

Wie Sie sehen, wird in diesem Programm sowohl die sequentielle als auch die wahlfreie Dateiform verwendet. Die Feldnamen und die Länge der Felder wird sequentiell abgespeichert bzw. geladen (name.IDX), die Datensätze selbst in eine RANDOM-ACCESS-Datei gelegt (name.DAT). Man könnte natürlich auch bei kleinen Datenbanken und einem so großen Arbeits-Speicher wie beim ATARI ST alle Daten sequentiell in ein entsprechendes Datenfeld (String-Array) einladen und direkt im Speicher verwalten.

Dies kostet aber Ladezeit und funktioniert nur, wenn nach der Manipulation der Daten auch alles wieder abgespeichert wird (was man ja vergessen könnte...). Außerdem wurde diese Form gewählt, um die Anwendung der wahlfreien Dateiform zu demonstrieren. Ich hoffe, Sie können mit diesem Programm durch Verändern und Erweitern genau das Datenbank-Programm erstellen, welches Ihre speziellen Anforderungen erfüllt!

(Feld-Nummer 1) den Text 'Bastelbude' enthält, angezeigt. Sie können hier auch nur Bas als Suchstring eingeben, da es wohl kaum noch andere Ortsnamen mit diesem Anfang gibt.

[illegible] Ausgabe von Daten

Diese Funktion ermöglicht die Ausgabe aller Datensätze auf dem Bildschirm oder dem Drucker. Die Abfrage, wohin ausgegeben werden soll, wird für das Drucken mit D oder d beantwortet, alle anderen Eingaben bewirken die Bildschirmausgabe.

Die Ausgabe erfolgt in der Reihenfolge, in der die Datensätze eingegeben wurden, es sei denn, Sie haben vorher die Sortier-Funktion aufgerufen.

[illegible] Ende

Hier wird nur der geöffnete Datenkanal geschlossen (CLOSE #1) und nach einer Abschiedsmeldung das Programm beendet.

Wie Sie sehen, wird in diesem Programm sowohl die sequentielle als auch die wahlfreie Dateiform verwendet. Die Feldnamen und die Länge der Felder wird sequentiell abgespeichert bzw. gelesen (name.IDX), die Datensätze selbst in eine RANDOM-ACCESS-Datei abgelegt (name.DAT). Man könnte natürlich auch bei kleinen Datenbanken und einem so großen Arbeits-Speicher wie beim ATARI ST die Daten sequentiell in ein entsprechendes Feld (String-Array) einladen und direkt im Speicher verwalten.

Das kostet aber Ladezeit und funktioniert nur, wenn nach der Manipulation der Daten auch alles wieder abgespeichert wird (was man ja vergessen könnte...). Außerdem wurde diese Form gewählt, um die Anwendung der wahlfreien Dateiform an [illegible]. Sie können mit diesem Programm durch Verändern und Erweitern genau das Datenbank-Programm erstellen, welches Ihre speziellen Anforderungen erfüllt!

# 3. Datenstrukturen

Man nehme eine große Menge Daten und bringe sie auf Diskette. Das klingt sehr einfach, aber schon beim genaueren Betrachten dieses Vorhabens fallen einige Dinge auf, die dabei problematisch sind.

Erst einmal muß die Verteilung auf der Diskette so erfolgen, daß man die Daten jederzeit wiederfinden kann. Dazu sind einige Vorbereitungen erforderlich, um die sich der Computerbenutzer zwar kaum kümmern muß, die aber das Betriebssystem und die Hardware des Rechners bzw. der Diskettenstationen und Festplatte übernehmen muß.

Die Diskette muß vor der Benutzung erst einmal formatiert werden. Dabei wird die Fläche der Disketten in einzelne Sektoren unterteilt, deren Position durch das verwendete Format festgelegt werden.

Dieses Format muß nun auch für den Rechner feststellbar sein, da er ja mit verschiedenen Formaten arbeiten können muß. Dabei sind die Anzahl der verwendeten Seiten der Diskette ebenso wichtig wie die Anzahl der Sektoren und deren Länge. Diese Informationen enthält der sogenannte Boot-Sektor, den wir gleich unter die Lupe nehmen werden.

Danach müssen für jede Datei bzw. Programm, welches auf der Diskette gespeichert werden soll, die verwendeten Sektoren zugewiesen und markiert werden. Diese Informationen sind im Inhaltsverzeichnis der Diskette und im sogenannten FAT, der 'File-Allocation-Table' verborgen. Diese werden ebenfalls in diesem Kapitel besprochen.

Fangen wir also am Anfang an: bei der Formatierung.

## 3.1 Diskettenformat

Beim Formatieren einer Diskette wird diese, wie bereits erwähnt, in einzelne Abschnitte aufgeteilt. Die grobe Unterteilung ist die Spureinteilung. Diese Spuren liegen wie konzentrische Ringe auf der Scheibe und werden von außen nach innen gezählt. Auf einer normal formatierten Diskette befinden sich 80 solche Spuren, auch Tracks genannt, die von 0 bis 79 nummeriert sind. Es ist zwar möglich, bis zu 82 Tracks zu formatieren, jedoch nimmt die Datensicherheit zur Mitte hin wegen der immer geringeren verfügbaren Fläche dermaßen ab, daß die Tracks 80 bis 82 nicht mehr verwendet werden. Dennoch können sie verwendet werden, wenn man entsprechend formatiert.

Die einzelnen Tracks werden nun ihrerseits in Sektoren unterteilt, die je einen Abschnitt des Ringes darstellen. Diese Sektoren werden wiederum zu sogenannten Clustern zusammengefaßt, üblicherweise 2 Sektoren pro Cluster. Die Bedeutung dieser Cluster ist jedoch nicht so wichtig, so daß wir nur die Sektoren betrachten.

Im normalen Format befinden sich 9 Sektoren auf jedem Track, die 512 Bytes fassen. Somit ergibt sich bei einer einseitigen Diskette eine Speicherkapazität von 80*9*512=368640 Bytes.

Dies ist allerdings nicht die wirkliche Anzahl der auf der Diskette gespeicherten Daten. Pro Track und nochmals pro Sektor werden beim Formatieren einige zusätzliche Daten geschrieben.

Diese Daten werden vom Disk-Controller, dem die Floppy steuernden Chip im ST, benötigt, um aus dem Track den richtigen Sektor herauszufinden. Betrachten wir nun den vollständigen Aufbau eines normalen Tracks.

| Anzahl | Bytes | Bemerkungen |
|---|---|---|
| 60 | $4E | Track-Anfang |
| pro Sektor: | | |
| 12 | $00 | |
| 3 | $F5 | werden als $A1 geschrieben |
| 1 | $FE | ID Adress Mark |
| 1 | Track# | Tracknummer 0 bis 79 |
| 1 | Seiten# | Seitennummer 0 oder 1 |
| 1 | Sektor# | Sektornummer 1 bis 9 |
| 1 | $02 | *$100=512 Bytes pro Sektor |
| 1 | $F7 | CRC-Prüfsumme schreiben (werden 2 Bytes) |
| 22 | $4E | Füllbytes |
| 12 | $00 " | |
| 3 | $F5 | werden zu $A1 |
| 1 | $FB | Markierung (Data-Adress-Mark) |
| 512 | Daten | hier liegen die eigentlichen Daten |
| 1 | $F7 | CRC-Prüfsumme schreiben |
| 40 | $4E | Füllbytes |
| usw. | | |
| Trackende: | | |
| 1401 | $4E | Füllbytes |

Zählt man all diese Bytes zusammen, so kommt man auf 6969 Bytes pro Track, was einer (unformatierten) Diskettenkapazität von 557520 Bytes entspricht. Leider kann man diese Kapazität nicht für seine Daten nutzen, da der Controller sie dann nicht mehr finden könnte (wie könnte er Anfang und Ende eines Sektors erkennen?). Möglich ist es jedoch, die letzten 1401 Bytes jedes Tracks für einen zusätzlichen Sektor zu benutzen. Das würde die benutzbare Kapazität auf 409600 Bytes erhöhen. Nimmt man zusätzlich noch drei weitere Tracks (80 bis 82) dazu, kommt man sogar auf 424960 Bytes. Aber wie gesagt, die Datensicherheit ist nicht mehr so besonders, wenn auch vertretbar.

Um ein solches eigenes Format zu erstellen, ist ein kleines Programm nötig. Bevor wir uns ein solches Programm ansehen, müssen wir jedoch die einzelnen Schritte, in denen das Formatieren abläuft, genauer ansehen. Mit dem bloßen Formatieren der Tracks ist es nämlich nicht getan. Die verwendeten Parameter wie Anzahl der Tracks und Sektoren müssen ebenfalls auf die Diskette geschrieben werden, da der Rechner sonst nicht feststellen kann, wie die Diskette formatiert ist. Dazu dient der Boot-Sektor.

## 3.2 Der Boot-Sektor

Der Boot-Sektor liegt immer ganz am Anfang einer Diskette bzw. Festplatte, d.h. auf Track 0, Seite 0, Sektor 1 einer Diskette oder Sektor 0 der Harddisk. Er ist, wie alle anderen Sektoren auch, 512 Bytes lang und wird nach jedem Diskettenwechsel vom Betriebssystem überprüft. Außerdem ist er für das 'Booten' der Diskette entscheidend. Booten bedeutet hier das Laden des Betriebssystems von Diskette nach dem Einschalten. Dabei wird zuerst von der Diskette in Laufwerk A der Boot-Sektor geladen und geprüft, ob die Diskette ein Betriebssystem enthält. Außerdem enthält der Boot-Sektor noch weitere Informationen.

Insgesamt enthält der Boot-Sektor eine Seriennummer der Diskette, einen Parameterblock für das BIOS des Rechners und evtl. ein Boot-Programm mit Bootparametern. Sollte ein solches Programm enthalten sein, so muß die Summe aller enthaltenen Bytes des Sektors (Checksumme) die 'magische' Zahl $1234 ergeben. Stimmt diese Zahl, so wird das Programm ab dem Anfang des Sektors ausgeführt, wo dann üblicherweise ein BRA (Branch always)-Befehl steht. Das Programm muß dabei so gestaltet sein, daß es an jeder beliebigen Speicherstelle laufen kann, da der Sektor ja irgendwohin geladen wurde.

Solch ein Boot-Programm ist normalerweise nicht im Boot-Sektor enthalten. Wichtiger sind die verschiedenen Parameter, die sich ebenfalls auf dem Sektor befinden. Diese Parameter werden bei einem 'Get BPB'-Aufruf des Betriebssystems geladen und in den sogenannten BPB (BIOS-Parameter-Block) geladen. Sind

diese Parameter unglaubwürdig, so wird aus der 'Get BPB'-Funktion nicht die Adresse des BPB übergeben, sondern eine 0.

Die weitere Information im Boot-Sektor ist die Seriennummer der Diskette. Es handelt sich dabei um eine 24Bit-Zahl, die beim Formatieren ermittelt und auf die Diskette geschrieben wurde. Diese Nummer dient zur Erkennung eines Diskettenwechsels.

Hier nun der gesamte Aufbau des Boot-Sektors:

| Byte Nr. | Name | Bedeutung (normal 1/2-seitig) |
|---|---|---|
| $00 | BRA | Sprungbefehl in das Boot-Programm (evtl.) |
| $02 | Füller | reservierte Füllbytes oder 'Loader' |
| $08 | Serien# | Seriennummer |
| * $0B | BPS | Anzahl Bytes pro Sektor (512) |
| * $0D | SPC | " Sektoren pro Cluster (2) |
| * $0E | RES | " reservierte Sektoren (1) |
| * $10 | NFATS | " FATs (File Allocation Tables) (2) |
| * $11 | NDIRS | " mögliche Directory-Einträge (112) |
| * $13 | NSECTS | " Sektoren auf Diskette (720/1440) |
| * $15 | MEDIA | Medium-Beschreibung (unbenutzt) |
| * $16 | SPF | Anzahl Sektoren der FAT (5) |
| * $18 | SPT | " Sektoren pro Track (9) |
| * $1A | NSIDES | " Seiten der Diskette (1/2) |
| * $1C | NHID | " versteckte Sektoren (0) |
| $1E | EXECFLG | Flag für COMMAND.PRG |
| $20 | LDMODE | Flag für File- oder Sektorboot |
| $22 | SSECT | erster zu ladender Sektor |
| $24 | SECTCNT | Anzahl der zu ladenden Sektoren |
| $26 | LDADDR | Ladeadresse |
| $2A | FATBUF | FAT-Adresse |
| $2E | FNAME | Filename (meist TOS.IMG) |
| $39 | RES | reserviert |
| $3A<br>-$1FD | BOOTIT | Boot-Programm |
| $1FE | | Ausgleichswort für die Checksumme |

Die mit einem Sternchen (*) gekennzeichneten Einträge entsprechen dem BPB der Diskette. Diese Tabelle ist identisch mit derjenigen des MS-DOS, des Betriebssystems der IBM-PCs. Dadurch ist auch zu beachten, daß ein 16Bit-Wort hier in der Bytes-Reihenfolge low-high vorliegt (z.B. BPS= $00 $02 ergibt $200 Bytes pro Sektor). Somit ist der ATARI ST in der Lage, IBM-Disketten zu lesen. Er kann sie jedoch nicht ohne weiteres auswerten, da die Datenverteilung auf der Diskette anders organisiert ist als beim ATARI ST.

Hier noch einige Anmerkungen zu den Einträgen im Boot-Sektor:

> Die Zahlen in Klammern, die hinter einigen Einträgen stehen, stellen den üblichen Inhalt dieser Einträge bei einer einseitig formatierten Diskette dar.
>
> NHID, die Anzahl der versteckten Sektoren, wird vom BIOS des ATARI ST bei Disketten nicht verwendet.

Die ab $1E liegenden Daten sind nur interessant, wenn es sich bei der Diskette um eine Boot-fähige Diskette handelt. Eine solche Diskette enthält normalerweise das Betriebssystem in Form eines Datenfiles, Imagefile (.IMG) genannt. Man erkennt einen ausführbaren Boot-Sektor auch daran, daß in ihm ab dem 3.Byte der Text 'Loader' steht. Das Boot-Programm, welches in den älteren ATARI STs in den zwei ROMs steckt, erkennt einen solchen Boot-Sektor außerdem an der Prüfsumme, welche für einen ausführbaren Boot-Sektor $1234 betragen muß. Liegt also ein solcher Fall vor, bekommen die weiteren Daten im Boot-Sektor folgende Bedeutungen:

*EXECFLG* wird in die Systemvariable 'cmdload' kopiert. Dieses Flag entscheidet, ob nach dem Laden des Betriebssystems das Programm COMMAND.PRG geladen werden soll oder nicht.

*LDMODE* bestimmt den Lademodus. Ist dieses Flag Null, wird das mit FNAME benannte File gesucht und geladen. Dieses File

ist üblicherweise TOS.IMG. Ist LDMODE ungleich null, so werden Sektoren in Abhängigkeit von SECTCNT und SSECT direkt geladen.

*SSECT* ist der logische Sektor, ab dem gebootet wird. Diese Variable ist nur gültig, wenn LDMODE ungleich null ist.

*SECTCNT* gibt die Anzahl der zu bootenden Sektoren an. Dies ist ebenfalls nur bei LDMODE ungleich null gültig.

*LDADDR* ist die Speicheradresse, ab der das File oder die Sektoren geladen werden.

*FATBUF* gibt die Adresse an, an die die FAT- und die Directory-Sektoren geladen werden sollen.

*FNAME* ist der Filename des 'Image-Files', welches geladen werden soll (LDMODE = 0). Es ist genauso aufgebaut wie ein normaler Filename, also 8 Zeichen als Name und 3 Zeichen als Extender.

*BOOTIT* ist ein eventuelles Boot-Programm, welches nach dem Laden des Boot-Sektors ausgeführt wird.

So ist also der Boot-Sektor aufgebaut. Zusammen mit den Kenntnissen des Diskettenformates ist nun genug Theorie vorhanden, um einen Schritt in die Praxis zu tun. Wir wollen diesen Schritt mit einem Programm vornehmen, mit dem wir unsere Disketten formatieren können.

Mit Hilfe des 'File'-Menüs sind wir bereits in der Lage, Disketten zu formatieren. Wie jedoch bereits anfangs erwähnt, ist das vom ATARI-Betriebssystem TOS verwendete Format auf 80 Tracks und 9 Sektoren pro Tracks festgelegt. Physikalisch passen aber wesentlich mehr Tracks und Sektoren auf eine Diskette.

### 3.2.1 Ein Formatierungsprogramm

Das nun folgende Programm bietet einige Möglichkeiten, die Kapazität einer normalen Diskette zu erhöhen. Es wird über ein kleines Menü gesteuert, in dem man einige Parameter für die Formatierung einstellen kann. Die Auswahl der gewünschten Einstellungen erfolgt dabei über die Funktionstasten.

Das Menü, welches nach dem Aufruf des Programmes erscheint, sieht folgendermaßen aus:

```
*** Formatierungs-Programm S.D. ***

 [F1] Seite(n) ........: 2
 [F2] Tracks ..........: 80
 [F3] Sektoren/Track ..: 9
 [F4] Laufwerk ........: A
 [F8] Formatieren ...
[F10] Quit !
```

Durch Druck einer der angegebenen Funktionstasten wird nun entweder eine Einstellung verändert oder eine Funktion ausgelöst. Folgende Einstellungen sind möglich:

F1: Mit dieser Taste wird zwischen ein und zwei Seiten umgeschaltet. Bei Verwendung eines einseitigen Laufwerks sollte auch nur eine Seite eingestellt werden.

F2: Hier wird zwischen 80 (Normaleinstellung) und 82 Tracks ausgewählt. Die Verwendung von 83 Tracks wäre auch möglich, aus Gründen der zu geringen Datensicherheit habe ich jedoch auf diese Auswahlmöglichkeit verzichtet. Sie können natürlich dennoch durch geringfügige Änderungen im Programm mit 83 Tracks arbeiten.

F3: Mit der Funktionstaste F3 wird zwischen 9 und 10 Sektoren pro Track hin- und hergeschaltet.

F4: Hiermit können Sie zwischen Laufwerk A und B auswählen. Dieser Punkt ist vor dem Starten der Formatierung besonders zu kontrollieren, damit nicht versehentlich Ihre im anderen Laufwerk steckende Systemdiskette formatiert wird...

F8: Diese Taste löst die Formatierung selbst aus. Dieser Vorgang beginnt unmittelbar auf Tastendruck und wird durch die Meldung "Formatierung läuft. Bitte warten..." angezeigt. Sollte ein Fehler auftauchen, erscheint die Meldung "** Es ist ein Fehler aufgetreten !! **". Sie sollten daraufhin die Diskette überprüfen, ob nicht z.B. der Schreibschutz zurückgeschoben ist. Die Fehlermeldung bleibt solange auf dem Bildschirm, bis irgendeine Taste gedrückt wird. Somit braucht der Formatierungs-Vorgang nicht ständig beobachtet zu werden.

F10: Wenn Sie alle Disketten formatiert haben, können Sie durch die Betätigung der F10-Taste das Programm verlassen.

Durch die flexible Auswahlmöglichkeit dieses Programmes sind verschiedene Kapazitäten der formatierten Disketten einstellbar. Hier einige Werte für einseitige Formate:

| Tracks | Sektoren pro Track | Kapazität in Bytes |
|---|---|---|
| 80 | 9 (normal) | 357376 |
| 82 | 9 | 366592 |
| 80 | 10 | 398336 |
| 82 | 10 | 408576 |

Wie Sie an obiger Tabelle erkennen können, ist ein Gewinn an Kapazität schon bei einseitigen Disketten bis zu 51200 Bytes möglich. Bei doppelseitigen Disketten verdoppelt sich diese Ausbeute noch, so daß über 100 KByte dazukommt.

Hier nun das Programm. Erstellt wurde es mit dem Programm SEKA, welches geringe Abweichungen zum DRI-Assembler

zeigt. Sollten Sie das Programm mit dem DRI-Assembler übersetzen wollen, so müssen lediglich die Kommentarzeilen mit einem Sternchen (*) beginnen und die 'blk.b'-Anweisung in 'ds.b' geändert werden.

```
            ;** Formatierungs-Programm S.D. **
run:
            move.l  #menue,d0
            bsr     print           ;Menü ausgeben
            bsr     getkey
            cmp.b   #$3b,d0
            blt     run             ;falsche Taste
            cmp.b   #$44,d0
            bgt     run             ;falsche Taste

            cmp.b   #$3b,d0         ;F1 ?
            bne     notf1
            eor     #3,sds          ;1/2 Seiten
            eor     #1,sdsf
            bra     run

notf1:
            cmp.b   #$3c,d0         ;F2 ?
            bne     notf2
            eor     #2,trs          ;80/82 Tracks
            eor     #2,trsf
            bra     run

notf2:
            cmp.b   #$3d,d0         ;F3 ?
            bne     notf3
            eor     #3,sptf
            eor     #$1109,spt      ;9/10 Sektoren pro Track
            bra     run

notf3:
            cmp.b   #$3e,d0         ;F4 ?
            bne     notf4
```

```
        eor     #3,lw
        eor     #1,lwf             ;Laufwerk A/B
        bra     run

notf4:
        cmp.b   #$42,d0            ;F8 ?
        bne     notf8
        bsr     format             ;=> Formatierung
        bra     run

notf8:
        cmp.b   #$44,d0            ;F10 ?
        bne     run
        clr     -(sp)
        trap    #1                 ;Quit, zurück zum Desktop

format:                            ;* Formatieren *
        move.l  #wait,d0
        bsr     print              ;"Formatierung läuft.."
        move    trsf,trsf1
        subq    #1,trsf1

floop:
        move    sdsf,seite         ;Seite bestimmen
floop1:
        bsr     fmttr              ;formatiere einen Track
        bne     error
        subq    #1,seite           ;ggf. noch andere Seite
        bpl     floop1             ;formatieren
        subq    #1,trsf1
        bpl     floop              ;nächster Track

setboot:                           ;Boot-Sektor erstellen
        clr     -(sp)              ;Execute-Flag: nicht ausführbar
        moveq   #2,d0
        or      sdsf,d0
        move    d0,-(sp)           ;Disktyp- und Seitenauswahl
        move.l  #$1000000,-(sp)    ;Seriennr. erstellen
        pea     puffer             ;Puffer-Adresse
        move    #$12,-(sp)
```

```
        trap    #14             ;Boot-Sektor erstellen
        add.l   #14,sp

        lea     puffer,a0       ;Zeiger auf Boot-Sektor-Puffer
        clr.l   d0
        cmp     #9,sptf         ;9 Sektoren pro Track ?
        beq     sok             ;ja
        move.b  #10,24(a0,d0)   ;sonst 10 SPT einsetzen
        move    trsf,d1         ;Anzahl der Tracks in D1
        tst     sdsf            ;1 Seite ?
        beq     sd11            ;ja
        lsl     #1,d1           ;sonst doppelter Zuwachs
sd11:
        bsr     addsec          ;SEC + Anzahl der Tracks (D1)

sok:
        cmp     #80,trsf        ;80 Tracks ?
        beq     trok            ;ja
        move    #18,d1
        tst     sdsf            ;1 Seite ?
        beq     sd12            ;ja
        lsl     #1,d1           ;sonst doppelter Zuwachs
sd12:
        bsr     addsec          ;SEC + 2*9 oder 4*9

trok:
        move    #1,-(sp)        ;1 Sektor
        clr.l   -(sp)           ;Seite 0, Track 0
        move    #1,-(sp)        ;Sektor 1
        move    lwf,-(sp)       ;Laufwerk
        clr.l   -(sp)
        pea     puffer          ;Puffer
        move    #9,-(sp)
        trap    #14             ;flopwr, Boot-Sektor schreiben
        add.l   #20,sp
        tst     d0              ;Fehler aufgetreten ?
        bne     error           ;ja: Fehlermeldung

        bra     run             ;Neustart
```

```
addsec:                                 ;SEC = SEC + D1
        move.b  20(a0,d0),d2            ;HI
        lsl     #8,d2
        move.b  19(a0,d0),d2            ;LO
        add     d1,d2
        move.b  d2,19(a0,d0)            ;set LO
        lsr     #8,d2
        move.b  d2,20(a0,d0)            ;set HI
        rts

error:
        move.l  #errtxt,d0
        bsr     print                   ;Fehlermeldung ausgeben
        bsr     getkey                  ;auf Taste warten
        bra     run                     ;und Neustart

fmttr:                                  ;einen Track formatieren
        clr     -(sp)                   ;Virgin-Daten
        move.l  #$87654321,-(sp)        ;Magic-Zahl
        move    #1,-(sp)                ;interleave
        move    seite,-(sp)             ;Seite
        move    trsf1,-(sp)             ;Track
        move    sptf,-(sp)              ;Sektoren/Track
        move    lwf,-(sp)               ;Laufwerk
        clr.l   -(sp)
        pea     puffer                  ;Track-Puffer
        move    #10,-(sp)
        trap    #14                     ;flopfmt, Track formatieren
        add.l   #26,sp
        tst     d0                      ;Test auf Error
        rts

print:                                  ;Text ab (D0) ausgeben
        move.l  d0,-(sp)
        move    #9,-(sp)
        trap    #1
        addq.l  #6,sp
        rts

getkey:                                 ;auf Tastendruck warten,
```

```
        move.w  #1,-(sp)
        trap    #1
        addq.l  #2,sp
        swap    d0              ;Tastencode in D0.b
        rts

; Texte und Variablen:

menue:  dc.b  $1b,"E*** Formatierungs-Programm S.D. ***"
        dc.b  10,13,10,13
        dc.b  " [F1] Seite(n) ........: "
sds:    dc.b  " 2",10,13
        dc.b  " [F2] Tracks ..........: "
trs:    dc.b  "80",10,13
        dc.b  " [F3] Sektoren/Track ..: "
spt:    dc.b  " 9",10,13
        dc.b  " [F4] Laufwerk ........: "
lw:     dc.b  " A",10,13
        dc.b  " [F8] Formatieren ...",10,13
        dc.b  "[F10] Quit !",10,13,10,13,0

wait:   dc.b "Formatierung läuft. Bitte warten...",10,13,0
errtxt: dc.b "** Es ist ein Fehler aufgetreten !! **",10,13,0

even
sdsf:   dc.w 1
trsf:   dc.w 80
trsf1:  dc.w 80
sptf:   dc.w 9
lwf:    dc.w 0
seite:  dc.w 0

data
puffer: blk.b 8000
```

Das Programm ist in folgende Abschnitte eingeteilt:

1. Menüsteuerung: Das Menü wird ausgegeben (es löscht dabei den Bildschirm) und auf einen Tasten-

druck gewartet. Nach einer Eingabe wird der Tastencode, welcher in D0 übergeben wurde, ausgewertet. Trifft einer der CMP.B #$xx,D0-Vergleiche zu, so wird die gewählte Funktion ausgelöst. Bei einer Umschalt-Funktion (F1-F4) wird durch den EOR-Befehl diese Umschaltung sowohl im Menütext als auch in der entsprechenden Parameterzelle vorgenommen. Nach einer erfolgten Umschaltung wird wieder zum Start (run) gesprungen, außer bei der Taste F10, die mittels der TERM-Funktion des GEMDOS das Programm beendet.

2. Formatierung: Nach Ausgabe der Meldung "Formatierung läuft..." wird die Diskette ab dem eingestellten maximalen Track-1 bis Track 0 formatiert. Ist zweiseitige Formatierung gewählt, so wird jeder Track erst auf Seite 1 (Rückseite) und dann auf Seite 0 formatiert.

3. Erstellung des Boot-Sektors: Zuerst wird ein normaler Boot-Sektor über die 'protobt'-Funktion des XBIOS erzeugt. Dabei wird lediglich die Anzahl der Seiten berücksichtigt.

4. Korrektur des Boot-Sektors: Sind von der Norm abweichende Einstellungen vorhanden (10 Sektoren pro Track, 82 Tracks), so wird der erstellte Boot-Sektor entsprechend korrigiert. Dazu wird zunächst die Anzahl der Sektoren pro Track getestet. Ist sie 10, so wird dies zuerst in die SPT-Zelle des Boot-Sektors eingesetzt und dann die Anzahl der Tracks zur Anzahl der Sektoren auf der Diskette addiert. Danach wird die gewählte Track-Anzahl getestet und bei Bedarf die zusätzliche Sektorenanzahl aufaddiert.

5. Abspeichern des Boot-Sektors: Mit Hilfe der 'flopwr'-Funktion des XBIOS wird der aufbereitete neue Boot-Sektor auf Seite 0, Track 0, Sektor 1 geschrieben. Sollte ein Fehler auftreten, so wird dieser angezeigt.

6. Datenbereich: Hier sind die Texte des Menüs bzw. der Meldungen und die Variablen untergebracht. Der Puffer wird zwar in der Länge eingestellt, jedoch nicht auf die Diskette geschrieben, da er im '.bss'-Bereich liegt. Bei Verwendung des DRI-Assemblers muß daher hier statt 'data' ein '.bss' eingesetzt werden.

Hier nun ein BASIC-Programm, welches das Formatierungs-Programm mit dem Namen 'Bigformat.prg' auf der Diskette erstellt:

```
 10 '*** Erstellung des BIGFORMAT-Programms ***
 15 '
 20 ?:fullw 2:clearw 2:gotoxy 0,0
 25 ? "File >> bigformat.prg << wird erzeugt":?:?:?
 30 dim c%( 441):cs#=0
 35 for i=0 to 441
 40 read a$:c%(i)=val("&H"+a$)
 45 check#=check#+(c%(i))
 50 next i
 55 if check#= 4376703 then 70
 60 ?"Geht leider noch nicht, da etwas mit den DATA's nicht stimmt."
 65 goto 80
 70 bsave "bigformat.prg",varptr(c%(0)), 883
 75 ? "Das Programm >> bigformat.prg << ist nun geschrieben."
 80 ?:?:?:?"Bitte Taste drücken":a=inp(2):end
 85 '
 90 '********* DATA für bigformat.prg *********
 95 '
100 DATA 601A,0000,0332,0000,0000,0000,0000,0000
101 DATA 0000,0000,0000,0000,0000,0000,203C,0000
102 DATA 020A,6100,01EA,6100,01F2,0C00,003B,6D00
103 DATA FFEC,0C00,0044,6E00,FFE4,0C00,003B,6600
104 DATA 0016,0A79,0003,0000,024E,0A79,0001,0000
105 DATA 0326,6000,FFC8,0C00,003C,6600,0016,0A79
106 DATA 0002,0000,026C,0A79,0002,0000,0328,6000
107 DATA FFAC,0C00,003D,6600,0016,0A79,0003,0000
108 DATA 032C,0A79,1109,0000,028A,6000,FF90,0C00
```

```
109 DATA 003E,6600,0016,0A79,0003,0000,02A8,0A79
110 DATA 0001,0000,032E,6000,FF74,0C00,0042,6600
111 DATA 000A,6100,0012,6000,FF64,0C00,0044,6600
112 DATA FF5C,4267,4E41,203C,0000,02D6,6100,0140
113 DATA 33F9,0000,0328,0000,032A,5379,0000,032A
114 DATA 33F9,0000,0326,0000,0330,6100,00E6,6600
115 DATA 00D0,5379,0000,0330,6A00,FFF0,5379,0000
116 DATA 032A,6A00,FFDC,4267,7002,8079,0000,0326
117 DATA 3F00,2F3C,0100,0000,4879,0000,0332,3F3C
118 DATA 0012,4E4E,DFFC,0000,000E,41F9,0000,0332
119 DATA 4280,0C79,0009,0000,032C,6700,001E,11BC
120 DATA 000A,0818,3239,0000,0328,4A79,0000,0326
121 DATA 6700,0004,E349,6100,0050,0C79,0050,0000
122 DATA 0328,6700,0016,323C,0012,4A79,0000,0326
123 DATA 6700,0004,E349,6100,0030,3F3C,0001,42A7
124 DATA 3F3C,0001,3F39,0000,032E,42A7,4879,0000
125 DATA 0332,3F3C,0009,4E4E,DFFC,0000,0014,4A40
126 DATA 6600,001E,6000,FE76,1430,0814,E14A,1430
127 DATA 0813,D441,1182,0813,E04A,1182,0814,4E75
128 DATA 203C,0000,02FC,6100,0046,6100,004E,6000
129 DATA FE4C,4267,2F3C,8765,4321,3F3C,0001,3F39
130 DATA 0000,0330,3F39,0000,032A,3F39,0000,032C
131 DATA 3F39,0000,032E,42A7,4879,0000,0332,3F3C
132 DATA 000A,4E4E,DFFC,0000,001A,4A40,4E75,2F00
133 DATA 3F3C,0009,4E41,5C8F,4E75,3F3C,0001,4E41
134 DATA 548F,4840,4E75,1B45,2A2A,2A20,466F,726D
135 DATA 6174,6965,7275,6E67,732D,5072,6F67,7261
136 DATA 6D6D,2020,532E,442E,202A,2A2A,0A0D,0A0D
137 DATA 205B,4631,5D20,2053,6569,7465,286E,2920
138 DATA 2E2E,2E2E,2E2E,2E2E,3A20,2032,0A0D,205B
139 DATA 4632,5D20,2054,7261,636B,7320,2E2E,2E2E
140 DATA 2E2E,2E2E,2E2E,3A20,3830,0A0D,205B,4633
141 DATA 5D20,2053,656B,746F,7265,6E2F,5472,6163
142 DATA 6B20,2E2E,3A20,2039,0A0D,205B,4634,5D20
143 DATA 204C,6175,6677,6572,6B20,2E2E,2E2E,2E2E
144 DATA 2E2E,3A20,2041,0A0D,205B,4638,5D20,2046
145 DATA 6F72,6D61,7469,6572,656E,202E,2E2E,0A0D
146 DATA 5B46,3130,5D20,2051,7569,7420,210A,0D0A
147 DATA 0D00,466F,726D,6174,6965,7275,6E67,206C
148 DATA 8475,6674,2E20,4269,7474,6520,7761,7274
```

```
149 DATA 656E,2E2E,2E0A,0D00,2A2A,2045,7320,6973
150 DATA 7420,6569,6E20,4665,686C,6572,2061,7566
151 DATA 6765,7472,6574,656E,2021,2120,2A2A,0A0D
152 DATA 0000,0001,0050,0050,0009,0000,0000,0000
153 DATA 0002,2808,1408,1408,1408,260A,0406,0604
154 DATA 0E0A,0E0E,120A,1006,120E,1A08,341E,0606
155 DATA 0608,0006
```

Nun noch einige Bemerkungen zum Programm:

- Das Kopieren einer normalen auf eine erweiterte Diskette ist nur File für File möglich, da das Betriebssystem wegen der unterschiedlichen Diskettenformate nicht die Disketten direkt kopiert.
- Die Verwendung einer erweiterten Diskette als TOS-Systemdiskette ist nicht ohne weiteres möglich, da der Loader im Boot-Sektor fehlt. Daher muß für das Booten dieser Diskette der Boot-Sektor einer anderen Systemdiskette kopiert werden und danach die Einstellungen der erweiterten Diskette mit einem geeigneten Disketten-Monitor wieder eingegeben werden.
- Verwenden Sie die erweiterten Disketten nicht unbedingt als Träger von sehr wichtigen und einmaligen Daten. Sollte die von Ihnen verwendete Diskettemarke nämlich nicht allzuviel taugen, so kann schon mal der eine oder andere Sektor in den obersten Tracks ausfallen...

Doch nun zurück zur leidigen Theorie. Wie bereits erwähnt, wird aus den diversen Informationen der BIOS-Parameter-Block, kurz BPB, erstellt. Sehen wir uns jetzt diesen BPB etwas genauer an.

### 3.2.2 Der BIOS-Parameter-Block BPB

Einige Einträge dieses Parameterblocks sind uns ja schon bekannt, da sie auch im Boot-Sektor auftauchen. Der BPB wird beim Aufruf des BIOS-Kommandos 'Get BPB' (Nr. 7) erstellt, wenn die Diskette zwischenzeitlich gewechselt wurde. Der BPB enthält im Gegensatz zum Boot-Sektor seine Daten im normalen 16 Bit-Format, und zwar in folgender Reihenfolge:

| | | |
|---|---|---|
| recsize | Sektorgröße in Bytes | (512) |
| clziz | Clustergröße in Sektoren | (2) |
| clsizb | Clustergröße in Bytes | (1024) |
| rdlen | Anzahl der Directorysektoren | (7) |
| fsiz | FAT-Größe in Sektoren | (5) |
| fatrec | Startsektor des zweiten FATs | (6) |
| datrec | erster Datensektor | (rdlen+fsiz+fatrec=18) |
| numcl | Anzahl der Datencluster | (711) |
| bflags | FAT-Eintragsgröße in Bit 0: | |
| | 0=12 Bit, 1=16 Bit | (0) |

Die Zahlen in Klammern geben den typischen Inhalt der Einträge bei einer doppelseitig formatierten Diskette an.

Nun wollen wir wieder ein Programm betrachten, mit dem dieser BIOS-Parameter-Block BPB eingelesen und analysiert werden kann. Das Programm ist recht einfach aufgebaut. Zuerst wird ein Prompt ausgegeben, welches auch die Überschrift beinhaltet. Dieses Prompt fordert nun zur Eingabe eines Buchstabens in die Tastatur auf. Dabei ist entweder eine Laufwerksbezeichnung (a,b,c oder d) oder der Buchstabe 'q' gültig. 'q' beendet das Programm und führt wieder zum Desktop zurück.

Nach der Eingabe testet das Programm, ob ein gültiger Buchstabe eingegeben wurde. Wenn nicht, so wird erneut gestartet,

wenn ein 'q' eingegeben wurde, so wird das Programm abgebrochen.

Der Buchstabe wird dann durch Subtraktion von 'a' in die für den GETBPB-Aufruf nötigen Wert (0-3) umgewandelt. Damit wird nun die GETBPB-Funktion aufgerufen. Man erhält die Adresse des BPB im Register D0 zurück.

Die Einträge des BPB werden nun nacheinander ausgelesen, sedezimal (= hexadezimal) ausgegeben und mit dem entsprechenden Text versehen. Damit werden alle wichtigen Informationen über die Diskette auf einen Blick überschaubar.

Hier das Programm, welches wiederum mit dem SEKA geschrieben wurde:

```
;** BPB-Analysator S.D. **
run:
        move.l  #prompt,d0
        bsr     pmsg            ;Prompt ausgeben
        bsr     getkey          ;Eingabe des Laufwerks A-D
        cmp     #'q',d0         ;Quit ?
        beq     quit            ;ja => Desktop
        move    d0,d6           ;Zeichen retten
        bsr     pcrlf           ;CR ausgeben

        sub     #'a',d6         ;Wert umwandeln
        bmi     run             ;falsche Eingabe
        cmp     #3,d6
        bgt     run             ;falsche Eingabe

        move    d6,-(sp)        ;Device-Nr.
        move    #7,-(sp)
        trap    #13             ;GETBPB-Funktion
        addq.l  #4,sp

        tst.l   d0
        beq     run             ;Error !
```

```
        move.l  d0,a5           ;BPB-Adresse retten

        bsr     pnext
        move.l  #bps,d0
        bsr     pline           ;"Bytes pro Sektor"

        bsr     pnext
        move.l  #spc,d0
        bsr     pline           ;"Sektoren pro Cluster"

        bsr     pnext
        move.l  #bpc,d0
        bsr     pline           ;"Bytes pro Cluster"

        bsr     pnext
        move.l  #dirsec,d0
        bsr     pline           ;"Directory-Sektoren"

        bsr     pnext
        move.l  #fatsec,d0
        bsr     pline           ;"FAT-Sektoren"

        bsr     pnext
        move.l  #fat2s,d0
        bsr     pline           ;"Start-Sektor des 2. FAT"

        bsr     pnext
        move.l  #datsec,d0
        bsr     pline           ;"Start-Sektor der Daten"

        bsr     pnext
        move.l  #datc,d0
        bsr     pline           ;"Datencluster"

        move    #'$',d0
        bsr     pchar           ;"$" ausgeben
        move    #12,d0          ;12 Bit annehmen
        btst    #0,(a5)         ;richtig ?
        beq     bits12          ;ja
        move    #16,d0          ;sonst 16 Bit
```

```
bits12:
        bsr     phexbyt
        move.l  #fatbit,d0
        bsr     pline           ;"Bits pro FAT-Eintrag"

        bra     run             ;fertig => Neustart

quit:                           ; Exit zum Desktop
        clr     -(sp)
        trap    #1

getkey:                         ;Get Key -> D0
        move    #1,-(sp)
        trap    #1
        and.l   #$ff,d0
        addq.l  #2,sp
        rts

pline:                          ;Print Line/CR
        bsr     pmsg
pcrlf:                          ;Print CR,LF
        move    #10,d0
        bsr     pchar
        move    #13,d0
pchar:                          ;Print Character D0
        move    d0,-(sp)
        move    #2,-(sp)
        trap    #1
        addq.l  #4,sp
        rts

pmsg:                           ;Print Line (D0)
        move.l  d0,-(sp)
        move    #9,-(sp)
        trap    #1
        addq    #6,sp
        rts

pnext:                          ;nächstes Wort holen und ausgeben
        move    #'$',d0
```

```
        bsr     pchar               ;"$" ausgeben
        move    (a5)+,d0
phexword:                           ;Print Hex-Word D0
        moveq   #3,d1
        bra     phex1
phexbyt:                            ;Print Hex-Byte
        moveq   #1,d1
        rol.l   #8,d0
phex1:
        rol.l   #4,d0
        move.l  d0,-(sp)
        move.l  d1,-(sp)
        bsr     phexnib             ;ein Nibble (0-F) ausgeben
        move.l  (sp)+,d1
        move.l  (sp)+,d0
        dbra    d1,phex1
        rts

phexnib:
        and.l   #$7f,d6
        swap    d0
        and.l   #$0f,d0
        add.b   #$30,d0
        cmp.b   #$3a,d0
        bcs     phexn
        add.b   #7,d0
phexn:
        bra     pchar               ;Nibble ausgeben


prompt:         dc.b "*** BPB-Analysator S.D. ***",10,13
                dc.b "Bitte Laufwerk eingeben (a-d) oder",10,13
                dc.b "'q' für Quit : ",0
bps:            dc.b " Bytes pro Sektor",0
spc:            dc.b " Sektoren pro Cluster",0
bpc:            dc.b " Bytes pro Cluster",0
dirsec:         dc.b " Directory-Sektoren",0
fatsec:         dc.b " FAT-Sektoren",0
fat2s:          dc.b ": Start-Sektor 2.FAT",0
datsec:         dc.b ": Start-Sektor der Daten",0
```

```
datc:        dc.b " Daten-Cluster",0
fatbit:      dc.b " Bits pro FAT-Eintrag",10,13,0
```

Hier auch gleich den BASIC-Lader, welcher das BPB-Analyseprogramm als BPBANA.TOS auf der Diskette erstellt:

```
10  '*** Erstellung des BPB-Analysators ***
15  '
20  ?:fullw 2:clearw 2:gotoxy 0,0
25  ? "File >> BPBANA.TOS << wird erzeugt":?:?:?
30  dim c%( 331):cs#=0
35  for i=0 to 331
40  read a$:c%(i)=val("&H"+a$)
45  check#=check#+(c%(i))
50  next i
55  if check#= 4548987 then 70
60  ?"Geht leider noch nicht, etwas mit den DATAs stimmt nicht."
65  goto 80
70  bsave "BPBANA.TOS",varptr(c%(0)), 664
75  ? "Das Programm >> BPBANA.TOS << ist nun geschrieben."
80  ?:?:?:?"Bitte Taste drücken":a=inp(2):end
85  '
90  '********* DATAs für BPBANA.TOS *********
95  '
100 DATA 601A,0000,026E,0000,0000,0000,0000,0000
101 DATA 0000,0000,0000,0000,0000,0000,203C,0000
102 DATA 015E,6100,0100,6100,00D0,0C40,0071,6700
103 DATA 00C4,3C00,6100,00D6,0446,0061,6B00,FFDE
104 DATA 0C46,0003,6E00,FFD6,3F06,3F3C,0007,4E4D
105 DATA 588F,4A80,6700,FFC6,2A40,6100,00D4,203C
106 DATA 0000,01B2,6100,00A2,6100,00C6,203C,0000
107 DATA 01C5,6100,0094,6100,00B8,203C,0000,01DC
108 DATA 6100,0086,6100,00AA,203C,0000,01F0,6100
109 DATA 0078,6100,009C,203C,0000,0205,6100,006A
110 DATA 6100,008E,203C,0000,0214,6100,005C,6100
111 DATA 0080,203C,0000,0229,6100,004E,6100,0072
112 DATA 203C,0000,0242,6100,0040,303C,0024,6100
113 DATA 0048,303C,000C,0815,0000,6700,0006,303C
```

```
114 DATA 0012,6100,005C,203C,0000,0252,6100,001A
115 DATA 6000,FF2A,4267,4E41,3F3C,0001,4E41,0280
116 DATA 0000,00FF,548F,4E75,6100,001A,303C,000A
117 DATA 6100,0006,303C,000D,3F00,3F3C,0002,4E41
118 DATA 588F,4E75,2F00,3F3C,0009,4E41,5C4F,4E75
119 DATA 303C,0024,6100,FFE2,301D,7203,6000,0006
120 DATA 7201,E198,E998,2F00,2F01,6100,000C,221F
121 DATA 201F,51C9,FFF0,4E75,0286,0000,007F,4840
122 DATA 0280,0000,000F,0600,0030,0C00,003A,6500
123 DATA 0006,0600,0007,6000,FFA0,2A2A,2A20,2042
124 DATA 5042,2D41,6E61,6C79,7361,746F,7220,2053
125 DATA 2E44,2E20,202A,2A2A,0A0D,4269,7474,6520
126 DATA 4C61,7566,7765,726B,2065,696E,6765,6265
127 DATA 6E20,2861,2D64,2920,6F64,6572,0A0D,2771
128 DATA 2720,6681,7220,5175,6974,203A,2000,2020
129 DATA 4279,7465,7320,7072,6F20,5365,6B74,6F72
130 DATA 0020,2053,656B,746F,7265,6E20,7072,6F20
131 DATA 436C,7573,7465,7200,2020,4279,7465,7320
132 DATA 7072,6F20,436C,7573,7465,7200,2020,4469
133 DATA 7265,6374,6F72,792D,5365,6B74,6F72,656E
134 DATA 0020,2046,4154,2D53,656B,746F,7265,6E00
135 DATA 3A20,5374,6172,742D,5365,6B74,6F72,2032
136 DATA 2E46,4154,003A,2053,7461,7274,2D53,656B
137 DATA 746F,7220,6465,7220,4461,7465,6E00,2020
138 DATA 4461,7465,6E2D,436C,7573,7465,7200,2020
139 DATA 2020,4269,7473,2070,726F,2046,4154,2D45
140 DATA 696E,7472,6167,0A0D,0000,0000,0002,420E
141 DATA 0E0E,0E0E,0E0E,2600
```

Beim Einschalten des Rechners sind die Daten des BPB nicht vorhanden. Das Betriebssystem erstellt die BPB erst nach dem Booten, wenn es die Anzahl und Kennung der angeschlossenen Laufwerke feststellt. Doch dazu muß erst einmal ein Betriebssystem vorhanden sein.

Ist das TOS nicht im Rechner eingebaut, so muß es erst gebootet werden. Ebenso wird gebootet, wenn zwar ein Betriebssystem eingebaut ist, die Diskette jedoch ein bootbares Betriebssystem

(TOS.IMG) enthält und der Boot-Sektor ausführbar ist. Der Vorgang des Bootens läuft in folgenden 4 Schritten ab:

1. Der Boot-Sektor wird geladen und das auf ihm befindliche Boot-Programm wird ausgeführt.

2. Die FAT und das Directory wird von der aktuellen Diskette geladen. Der Lader sucht nun nach dem angegebenen Filenamen (meist TOS.IMG). Findet er es nicht, so gibt er eine Fehlermeldung zurück.

3. TOS.IMG wird ab der Speicheradresse $40000 geladen.

4. Das geladene Programm wird am Anfang gestartet.

Das TOS.IMG besteht nun seinerseits aus drei Teilen:

- ein Relocator, ein Programm, welches das Betriebssystem an die eigentlich vorgesehene Adresse ($6100) schiebt. Dieses Programm löscht den Bildschirm, schiebt den TOS- Imageblock an seine Heimatadresse und startet es dort.

- die Daten des Betriebssystems (BIOS, XBIOS)

- die Daten des GEM und des Desktop-Programms

Wie man also sieht, ist der Aufbau des Betriebssystems im File TOS.IMG recht kompliziert. Das eingebaute TOS, welches in den 6 PROMs (Programmable Read Only Memory) liegt, ist natürlich etwas kürzer, da es nur das Betriebssystem mit GEM enthält und keinen Relocator.

Gehen wir nun über in den nächsten Abschnitt der Datenstrukturen auf Disketten, und zwar dem Aufbau und der Verwaltung des Inhaltsverzeichnisses.

### 3.3 Das Inhaltsverzeichnis

Das Inhaltsverzeichnis beginnt auf einseitig formatierten Disketten auf Track 1, Sektor 3, und belegt 7 Sektoren einer einseitigen Diskette. Es enthält pro Eintrag außer dem Filenamen und der Extension noch eine Reihe weitere Daten, die für die Verwaltung der Diskette mehr oder weniger wichtig sind.

Jeder Eintrag im Inhaltsverzeichnis besteht aus 32 Bytes, die alle Informationen über das File enthalten, die das Betriebssystem benötigt. Diese 32 Bytes unterteilen sich in 8 Datenfelder, die folgendermaßen aufgebaut sind:

| | | |
|---|---|---|
| 1 | - Filename | 8 Bytes |
| 2 | - Filetyp (Extender) | 3 Bytes |
| 3 | - Attribut | 1 Bytes |
| 4 | - Reserviert | 10 Bytes |
| 5 | - Uhrzeit | 2 Bytes |
| 6 | - Datum | 2 Bytes |
| 7 | - erster Cluster | 2 Bytes |
| 8 | - File-Größe | 4 Bytes |

Das erste Feld enthält also den Filenamen. Dieser Name besteht aus ASCII-Zeichen, also nur Buchstaben und Ziffern. Dabei werden auch nur Großbuchstaben verwendet. Der Name ist auf 8 Zeichen begrenzt; hat er weniger als 8 Zeichen, so wird der Rest mit Leerzeichen (Blanks) aufgefüllt.

Ist das erste Byte des Namens eine Null, so bedeutet dies, daß der Eintrag bisher nie benutzt wurde. Wurde das File bereits verwendet und wieder gelöscht, so findet sich hier eine 229 ($E5).

Ist das erste Zeichen des Namens ein Punkt (.), so steht dieser Eintrag für ein spezielles Unterverzeichnis, einen Ordner.

Das darauffolgende Feld enthält den Filetyp, auch Extender genannt. Dieser Typ ist auf 3 Buchstaben begrenzt (z.B. PRG, TOS, BAS usw.) und wird ebenfalls bei Bedarf mit Leerzeichen aufgefüllt. Auch hier werden nur Großbuchstaben verwendet.

Nun folgt das Byte des File-Attributes. Es enthält bitweise kodiert den Status dieses Eintrages bzw. des Files. Die Bedeutung dieser Bits ist folgende:

| Bit | Bedeutung wenn gesetzt (1) |
|---|---|
| 0 | nur lesen erlaubt |
| 1 | verstecktes File |
| 2 | System-File |
| 3 | Eintrag ist Disketten-Name |
| 4 | Eintrag ist ein Ordner |
| 5 | File wurde geändert |

Nach diesem Byte folgen 10 Bytes, welche keine Bedeutung haben. Sie gelten lediglich als Reservebytes, die vielleicht für spätere Anwendungen verwendet werden sollen.

Nun folgen zwei Bytes, welche die Uhrzeit der letzten Modifikation des Files enthalten. Hierbei wurde zur Platzeinsparung eine spezielle Kodierung der Zeit verwendet.

Die 16 Bit des Uhrzeiteintrages teilen sich in 3 Sektionen, Stunden, Minuten und Sekunden. Diese Aufteilung sieht folgendermaßen aus:

Beispiel: 19:21:34 Uhr

| Stunde | Minuten | Sekunden/2 |
|---|---|---|
| 10011 | 010101 | 10001 |

Die Sekunden werden nur in Zweierschritten gewertet, daher steht in den unteren 5 Bits der Uhrzeit eine 17.

Das nächste Feld im Directory enthält das Datum der letzten Änderung der Datei. Die Aufteilung in Jahr, Monat und Tag geschieht hier auf ähnliche Weise wie bei der Uhrzeit. Dabei sind für das Jahr nur 7 Bits reserviert, weshalb grundsätzlich zu der hier enthaltenen Zahl 1980 addiert werden muß. Somit ergibt sich folgender Eintrag:

Beispiel: 12.05.1986

| Jahr | Monat | Tag |
|---|---|---|
| 0000110 | 0101 | 01100 |

Das siebente Feld im Directory enthält die Nummer des ersten Clusters auf der Diskette, der von dem File belegt wird. In diesem üblicherweise aus zwei Sektoren bestehenden Cluster beginnt somit die Speicherung des Files. Wie es ab dort weitergeht, erfahren Sie im nächsten Kapitel über die FAT.

Das letzte Feld enthält schließlich die Länge des Files in Bytes. Hierbei ist zu beachten, daß eventuell weniger Bytes gelesen werden als hier steht, was auch von der FAT abhängt. Die Filelänge ist somit nur als maximale Länge anzusehen.

Anhand dieser Kenntnisse über den Aufbau des Inhaltsverzeichnisses auf den Disketten sind Sie nun in der Lage, mit Hilfe eines Diskettenmonitors die Aufteilung der Diskette zu analysieren. Durch Änderungen der Werte können vielfältige Manipulationen vorgenommen werden, deren Wirkung jedoch manchmal unangenehm sein können. Aus diesem Grunde empfiehlt es sich, vor solchen Manipulationen eine Kopie der Diskette anzufertigen.

Will man nun ein Programm schreiben, in dem das Inhaltsverzeichnis einer Diskette ausgelesen werden soll, so muß man vor dem Aufruf der entsprechenden Funktion des Betriebssystems einen Puffer für die erwarteten Daten bereitstellen. Der Anfang dieses Puffers wird als die 'Disk Transfer Adress (DTA)' bezeichnet.

Dieser Puffer ist 44 Bytes lang und muß dem Betriebssystem durch den Aufruf einer besonderen Funktion angegeben werden. Danach kann die Suche nach Directory-Einträgen beginnen. Dafür wird mit der Funktion SFIRST (search first) der erste passende Eintrag, mit SNEXT (search next) die weiteren Einträge gesucht und in den Puffer an der DTA geladen werden.

Der Puffer enthält nach dem Aufruf alle Informationen, die auch im Directory-Fenster des Desktop erscheinen. Die Aufteilung der Daten ist folgende:

| Byte(s) | Inhalt |
|---|---|
| 0...20 | reserviert |
| 21 | File-Attribut |
| 22,23 | Uhrzeit |
| 24,25 | Datum |
| 26...29 | Filegröße in Bytes (LO,HI) |
| 30...43 | Filename und Extender |

Nun zum Programm. Diese Maschinenroutine setzt zuerst die DTA und sucht dann nach dem angegebenen Filenamen im Directory. Ist der angegebene Name nur '*.*', so wird der erste Eintrag, der dem vorgegebenen Attribut entspricht, geliefert. Ist überhaupt kein passender Eintrag vorhanden, so liefert die Funktion im Datenregister D0 die Fehlernummer -33, File nicht gefunden, zurück. Andernfalls ist dieses Register null.

```
MOVE.L   #PUFFER,-(SP)        * DTA übergeben
MOVE     #$1A,-(SP)           * SETDTA-Funktionsnummer
TRAP     #1                   * Betriebssystem aufrufen
ADDQ.L   #6,SP                * Stack reparieren
MOVE     #%11001,-(SP)        * Dateityp: alle Dateien
MOVE.L   #NAME,-(SP)          * Adresse des Filenamens
MOVE     #$4E,-(SP)           * SFIRST-Funktionsnummer
TRAP     #1                   * Betriebssystem aufrufen
ADDQ.L   #8,SP                * Stack reparieren
TST      D0                   * gefunden
BNE      WARNIX               * nein
```

```
usw.
.
.
PUFFER:    .ds.b 44          * Platz für die Daten
NAME:      .dc.b "*.*",0     * alle Namen erlaubt
```

Um danach den nächsten Eintrag zu suchen, genügt einfach der Programmteil:

```
MOVE       #$4F,-(SP)        * SNEXT-Funktionsnummer
TRAP       #1                * Betriebsystem aufrufen
ADDQ.L     #2,SP             * Stack reparieren
TST        D0                * gefunden
BNE        WARNIX            * nein, das war's wohl
```

Auf diese Weise läßt sich also leicht ein Programm schreiben, welches z.B. das Inhaltsverzeichnis einer Diskette auf dem Drucker ausgibt. Ein solches Programm, das ein komplettes Inhaltsverzeichnis inklusive der Ordner-Inhalte übersichtlich ausdruckt, finden Sie auch im Kapitel 5.3.

Läßt man sich nun im Desktop das Inhaltsverzeichnis der Diskette anzeigen, so werden Name, Extender, Datum, Uhrzeit und Länge der Files ausgegeben. Wenn Sie dann ein Programm anklicken, so muß das Betriebssystem nicht nur wissen, wo auf der Diskette das File beginnt, sondern auch, wo die weiteren Daten des Files stehen. Diese Informationen beinhaltet die FAT, die wir nun betrachten wollen.

## 3.4 Die FAT

Die FAT (File Allocation Table) belegt normalerweise 5 Sektoren auf einer (einseitigen) Diskette und beginnt normalerweise auf Track 0, Sektor 2 der Seite 0. Die Größe dieser Tabelle variiert je nach verwendetem Format. Sie wird verwendet, um die Verteilung jedes Files auf der Diskette zu speichern.

Der Grund dafür liegt darin, daß ein File nicht unbedingt Sektoren belegt, die direkt hintereinander liegen. Schließlich werden diejenigen Sektoren, die ein gelöschtes File beinhaltet hatten, wieder zur Speicherung neuer Daten freigegeben. Ein neues File, welches auf die Diskette geschrieben wird, wird auf solche freien Sektoren verteilt. Dabei werden belegte Sektoren einfach übersprungen.

Jeder Sektor muß also einen eigenen Eintrag in der FAT besitzen, um als frei oder belegt erkannt werden zu können. Um den Umfang der FAT geringer zu halten, werden immer zwei Sektoren zusammengefaßt und als Cluster bezeichnet, die von 2 bis zum Diskettenende durchnummeriert sind. Die FAT enthält somit nur noch einen Eintrag für zwei Sektoren.

Jeder Eintrag der FAT ist normalerweise 12 Bytes lang. Einige Formate verwenden 16Bit-Einträge, was wir jedoch hier vernachlässigen können. Durch die Aufteilung in 12 Bit ergibt sich, daß je zwei FAT-Einträge 3 Bytes einnehmen.

Die ersten zwei Einträge der FAT enthalten Format-Informationen, weshalb die Nummerierung auch erst bei 2 beginnt.

Jeder weitere Eintrag repräsentiert nun einen Cluster. Eine Null in einem Eintrag bedeutet, daß der entsprechende Cluster frei ist. Dies bedeutet natürlich nicht, daß die Sektoren keine Daten enthalten, da ein gelöschtes File nicht wirklich von der Diskette gelöscht wird. Das Löschen eines Files geschieht lediglich dadurch, daß im Inhaltsverzeichnis der erste Buchstabe des Namens durch eine $E5 ersetzt wird und die freiwerdenden Cluster durch eine Null in der FAT freigegeben werden. Die Daten selbst sind dennoch vorhanden, aber schwer zu finden.

Enthält ein FAT-Eintrag eine $FF7, so bedeutet dies einen unbrauchbaren Cluster. Solche Cluster werden beim Formatierenerkannt und markiert. Wenn solche Fehler auf der Diskette vorkommen, z.B. durch einen kleinen Kratzer auf der Diskette, erkennt man dies an der verminderten Kapazität, die nach dem Formatieren gemeldet wird. Sollte ein solcher Fehler jedoch in

Track 0 oder 1 auftauchen, so ist die Diskette unbrauchbar, da dort der Boot-Sektor, die FAT und das Directory liegen müssen.

Soll nun ein File geladen werden, so entnimmt das Betriebssystem dem Directory die Nummer des ersten Clusters, der die gewünschten Daten enthält. Der FAT-Eintrag dieses Clusters enthält nun seinerseits die Nummer des nächsten Clusters des Files. Dessen FAT-Eintrag enthält dann wieder die nächste Nummer und so weiter, bis ein Eintrag eine $FFF enthält. Dies bedeutet, daß der Cluster der letzte des Files ist.

Auch in der FAT können mittels eines Diskettenmonitors Änderungen vorgenommen werden. Hierbei ist die Wahrscheinlichkeit eines Datenverlustes jedoch so hoch, daß man unbedingt vorher eine Kopie der Diskette anfertigen sollte.

## 3.5 Programmaufbau

Der ATARI ST besitzt einen recht großen Speicher, in den mehrere Programme passen. In der Tat ist es möglich, mehrere Programme gleichzeitig in den Speicher zu legen und ablaufen zu lassen. Ein einfaches Beispiel hierfür sind die Accessories, die ja im Hintergrund laufen.

Diese offene Speicherzuteilung wirft allerdings ein Problem auf. Von den 8 Bit-Rechnern ist man gewöhnt, daß ein Programm (Maschinenprogramm) an einer ganz bestimmten Stelle im Speicher liegen muß, damit es auch läuft. Das liegt daran, daß Maschinenprogramme den Speicher direkt adressieren bzw. zu bestimmten Adressen verzweigen, an denen dann das entsprechende Teilprogramm liegen müßte.

Beim ATARI ST ist das jedoch nicht möglich. Woher sollte ein Programmierer auch im voraus wissen, wohin sein Programm geladen wird und ob dort nicht schon ein anderes Programm liegt?

Ein anderes Problem ist es, daß das Betriebssystem wissen muß, wie groß das geladene Programm ist und wieviel Speicher es

zum Laufen benötigt. Braucht das Programm nämlich einen zusätzlichen Speicher zum Abspeichern von z.B. eingegebenen Texten, so darf dieser Speicher nicht beim dazuladen eines anderen Programmes überschrieben werden.

Wie man sieht, reicht es nicht aus, wenn eine Programmdatei auf Diskette nur die Programmdaten selbst enthält. Der Aufbau einer solchen Datei soll in diesem Kapitel erläutert werden.

Ein lauffähiges Programm auf Diskette, also .PRG-, .TOS- und .TTP-Dateien, ist in 4 Abschnitte unterteilt. Diese Abschnitte sind die folgenden:

Anfang der Datei: File-Header
Programm mit Datenfeld
Symbol-Tabelle (wenn eine existiert)
Relocation-Daten (wenn vorhanden)

Ende der Datei:

Betrachten wir zunächst den ersten Teil: den Header.

### 3.5.1 Der Programm-Header

Der Header ist ein 14 Worte langer Programmvorspann, der die Längen der einzelnen Segmente enthält. Der Aufbau des Headers ist folgender:

| Byte Nr. | Inhalt |
|---|---|
| $00,$01 | $601A, der Maschinenbefehl BRA *+$1A |
| $02-$05 | Länge des Programm-Segmentes (text) |
| $06-$09 | Länge des Daten-Segmentes (data) |
| $0A-$0D | Länge des Zusatzspeicher-Segmentes (bss) |
| $0E-$11 | Länge der Symboltabelle |
| $12-$1B | 00, reserviert |

Der erste Eintrag ist ein Maschinenbefehl, der den Programmablauf zum Anfang des Programm-Segmentes verzweigt.

Es folgt die Länge des Programm-Segmentes. Dieses Segment, allgemein 'text'-Segment genannt, enthält das Programm selbst. Alle Adressen, die das Programm verwendet, sind darin so abgelegt, daß der Programmanfang als Adresse 0 angenommen wird. Daten, die dieses Segment erhält, sind unverändert.

Der nächste Eintrag enthält die Länge des Daten-Segmentes, 'data'-Segment genannt. Dieses Segment muß unmittelbar im Anschluß an das Programm liegen. In einem Maschinenprogramm wird mit einer 'data'-Anweisung die Trennung zwischen text- und data-Segment vorgenommen. Es handelt sich bei den Daten um initialisierte Daten, wie z.B. Texte oder Tabellen. Uninitialisierte Daten wie Datenpuffer für Diskettenoperationen oder Zwischenspeicher enthält das nächste Segment.

Die Länge dieses Zusatzspeichers enthält der vierte Eintrag des Headers. Diesen Speicherbereich nennt man 'bss'. Nach dem Laden des Programms wird dieser Speicherbereich dem Programm zur Verfügung gestellt und gleichzeitig für andere Anwendungen gesperrt. Sein Inhalt ist allerdings nicht definiert - er muß erst vom Programm beschrieben werden. Der Vorteil des bss-Segmentes gegenüber dem data-Segment ist der, daß dieser Bereich nicht in dem Diskettenfile enthalten sein braucht. Dadurch wird ein Programmfile nicht länger als nötig.

Eintrag Nummer fünf enthält die Länge der Symbol-Tabelle. Eine solche Tabelle ist selten vorhanden, da sie für die Funktion des Programms keine Rolle spielt. Eine Symbol-Tabelle wird, wenn vom Programmierer gewünscht, von einem Compiler bzw. einem Assembler an das compilierte bzw. assemblierte Programm angehängt. Die Symbole entsprechen dabei den in dem Quellprogramm verwendeten Labels von Routinen oder Daten. Der Vorteil einer solchen Tabelle ist z.B. für die Fehlersuche nicht zu verachten, da ein symbolischer Debugger wie der SID beim Disassembler-Listing zu jeder vom Programm verwendeten Adresse den symbolischen Namen anhängt. Ist die Test- und Fehlerphase

in der Programmentwicklung jedoch abgeschlossen, so empfiehlt es sich, die Symbol-Tabelle wegzulassen, da sie das Programmfile unnötig lang macht.

Jeder Eintrag in der Symbol-Tabelle ist 7 Worte lang und enthält den Namen, Typ und Wert des Labels:

| Byte | Inhalt |
|---|---|
| $0-$7 | Symbol-Name, endet mit einer Null |
| $8-$9 | Symbol-Typ, wie relocatible, global oder extern |
| $A-$C | Wert, wie Adresse, Register-Nr., Direktwert usw. |

Die gesamte Symboltabelle eines Programmfiles kann mit dem Programm NM68 ausgelesen und ggf. ausgedruckt werden. Dazu wird vom Command-Prompt aus eingegeben:

NM68 Filename

Durch Anhängen von >prn: kann die Ausgabe des NM68-Programms auch auf den Drucker umgeleitet werden, andernfalls erscheint das Ergebnis auf dem Bildschirm.

Zurück zum Aufbau des Programm-Headers. Die verbleibenden Bytes von $12 bis $1B sind reserviert für spätere Anwendungen, müssen jedoch null sein.

Auf den Header folgt nun direkt das Programm selbst, welches wie gesagt eigentlich nur an der Adresse $0000 funktionieren kann. Um es an der jeweiligen Adresse lauffähig zu machen, an die es geladen wurde, müssen nun alle absoluten Adressen, die in dem Programm auftauchen, geändert werden. Dazu braucht nur die wirkliche Startadresse zu den im Programm enthaltenen Adressen addiert werden. Aber woher weiß das Betriebssystem, das diese Änderungen ja vornehmen muß, wo die absoluten Adressen im Programm stehen? Die Antwort heißt Relocation-Tabelle.

### 3.5.2 Die Relocation-Tabelle

Hinter der Symboltabelle folgt die Relocation-Tabelle in der Programmdatei. Diese Tabelle enthält die Abstände zwischen den Langworten, die reloziert werden müssen. Das erste Langwort in dieser Tabelle gibt das erste zu ändernde Langwort ab dem Programmanfang an. Danach werden Bytes verwendet, deren Wert wiederum den Abstand zwischen dem gefundenen und dem nächsten zu ändernden Langwort angibt. Ist der Abstand zwischen zwei solchen Langworten größer als 254, so wird ein Byte mit dem Wert 1 eingesetzt und das so oft, bis wieder mit einem Wert kleiner als 255 das nächste betroffene Langwort gefunden werden kann.

Das erste Byte, das eine Null enthält, zeigt das Ende der Relocation-Tabelle an. An dieser Stelle endet auch die gesamte Programmdatei auf der Diskette.

Wird ein Programm nun geladen, so legt das Betriebssystem dieses Programm an eine freie Stelle im Speicher ab und reloziert es. Die Aufteilung des Programms im Speicher ist danach etwas anders als es vorher auf der Diskette war. Vor dem eigentlichen Programm, dem das data- und das bss-Segment folgt, liegt nämlich noch die sogenannte Basepage. Diese 256 Bytes lange Basepage stellt wieder einen Vorspann dar, der Informationen über die aktuelle wirkliche Aufteilung des Programms im Speicher enthält.

Die Basepage ist folgendermaßen organisiert:

| Byte | Länge | Inhalt |
|---|---|---|
| 00 | 4 | Startadresse des Arbeitsspeichers |
| 04 | 4 | HI-Adresse des Arbeitsspeichers +1 |
| 08 | 4 | Startadresse des Programms |
| 0C | 4 | Länge des Programm-Segmentes in Bytes |
| 10 | 4 | Startadresse des data-Segmentes |
| 14 | 4 | Länge des data-Segmentes in Bytes |
| 18 | 4 | Startadresse des bss-Segmentes |

| | | |
|---|---|---|
| 1C | 4 | Länge des bss-Segmentes in Bytes |
| 2C | 4 | Zeiger auf den 'Environment-String' |
| 80 | 80 | Text der Kommandozeile (z.B. bei .TTP) |

Alle ungenannten Einträge der Basepage sind reserviert. Nicht nur der Rechner benötigt die Daten aus dieser Tabelle. Ein Programm kann sie ebenfalls sehr gut brauchen. Das beste Beispiel dafür ist die Kommandozeile. Nimmt man für sein Programm den Typ .TTP, so gibt das Betriebssystem beim Aufruf des Programms ein Dialogfenster aus, in dem man Eintragungen machen kann. Diese Zeile kann das Programm nun auswerten.

Um an die Adresse der Kommandozeile heranzukommen, muß man in seinem Programm am Anfang etwa folgende Programmsequenz stehen haben:

```
run:  MOVE.L    4(SP),A0     ;Adresse der Basepage
      LEA       $80(A0),A0
```

A0 enthält nun die Adresse der Kommandozeile. Damit kann nun weitergearbeitet werden.

## 3.6 Festplattenformat

Wenden wir uns nun der Festplatte zu. Hier ist es wegen der enormen Speicherkapazität nicht so einfach gestaltet wie auf einer Diskette. Eine Festplatte ist nämlich in bis zu vier Bereiche aufgeteilt, von denen jeder einen Boot-Sektor enthält. Diese Bereiche werden 'Partitions' genannt.

Der erste Sektor auf der Harddisk (logischer Sektor 0) enthält die Informationen über die Aufteilung der Festplatte. Diese Informationen liegen wie folgt:

| Byte | Name | Bedeutung |
|---|---|---|
| $1C2 | hd_siz | Gesamtgröße der Harddisk in logischen Vektoren |
| $1C6 | p0_flg | Partition 0 existiert, wenn p0_flg >0<br>Ist Bit 7 gesetzt, wird hier gebootet |
| $1C7 | p0_id | Bezeichnung der Partition (GEM) |
| $1CA | p0_st | logische Sektornummer des ersten Sektors in der Partition |
| $1CE | p0_siz | Größe der Partition in Sektoren |
| $1D2 | p1_flg | |
| $1D3 | p1_id | s.o., Partition 1 |
| $1D6 | p1_st | |
| $1DA | p1_siz | |
| $1DE | p2_flg | |
| $1DF | p2_id | s.o., Partition 2 |
| $1E2 | p2_st | |
| $1E6 | p2_siz | |
| $1EA | p3_flg | |
| $1EB | p3_id | s.o., Partition 3 |
| $1EE | p3_st | |
| $1F2 | p3_siz | |
| $1F6 | bsl_st | Start-Sektor der 'bad sector list' |
| $1FA | bsl_cnt | Anzahl der defekten Sektoren |

Die 'bad sector list' wird beim Formatieren der Festplatte angelegt. Sie enthält eine Liste der defekten Sektoren, die sich nicht formatieren ließen. Die Tabelle liegt meist am Ende der Festplatte.

An der Variablen p*_flg erkennt das Betriebssystem, ob die Partition existiert (p*_flg ungleich Null). Der erste Sektor einer jeden Partition enthält einen Boot-Sektor, in dem die BPB liegt. Das Betriebssystem bootet von dem ersten Boot-Sektor, dessen p*_flg das Bit 7 gesetzt hat.

Ein Hinweis: Ein Programm zur Analyse und Anzeige der Partitions-Parameter finden Sie im Kapitel 5.1.1.4: Partitions-Analysator.

# 4. Die Diskettenlaufwerke

Die wohl bekannteste Art der Datenspeicherung ist die Verwendung von Disketten. Diese Speicherscheiben von $3\frac{1}{2}$ oder $5\frac{1}{4}$ Zoll Durchmesser (3 und 8 Zoll gibt es auch, sind aber für ATARI-Besitzer nicht wichtig) haben etliche Vorteile.

Da wäre erst einmal der Preis. Kostet eine $3\frac{1}{2}$-Zoll-Diskette z.B. 5 DM, entspricht das bei einer Kapazität von 360 KByte ungefähr 1,4 Pfennig pro Kbyte. Bei $5\frac{1}{4}$-Zoll-Disketten liegt dieses Verhältnis sogar noch günstiger. Da die Verwendung von $5\frac{1}{4}$-Zoll-Disketten technisch kein Problem für den ATARI ST bedeutet, ist dieser Preisvorteil ein Kriterium, welches für die Auswahl des zu verwendenden Diskettenformates eine Rolle spielt. Einige ATARI-Besitzer haben daher eine $3\frac{1}{2}$-Zoll- und eine $5\frac{1}{4}$-Zoll-Diskettenstation an ihren ST angeschlossen.

Ein weiterer Vorteil von Disketten gegenüber Festplatten ist die Wechselbarkeit. Somit kann auch mit nur einem Diskettenlaufwerk eine unbegrenzte Datenmenge verwaltet werden. Außerdem können Disketten zum Programm- und Datenaustausch hervorragend verwendet werden.

Doch nun muß auch der Haken bei der Sache erwähnt werden. Sieht man von der Speicherung auf Tonband-Kassetten einmal ab, sind Disketten die langsamsten Datenspeicher aus der heutigen Palette. Die Laufwerke der ATARI ST-Serie brauchen allerdings keinen Vergleich mit denen der Konkurrenz zu scheuen, da sie durch verschiedene technische Tricks im ATARI ST einen recht schnellen Datenaustausch ermöglichen.

Lassen Sie uns nun etwas tiefer in die Welt der Disketten eindringen.

### 4.1 Funktion

Werden vom Rechner Daten von der Diskette benötigt, so werden einige Funktionen innerhalb des Diskettenlaufwerks ausgelöst.

Als erstes wird der Motor des Laufwerks eingeschaltet. Hierbei fällt mehr oder weniger auf, daß bei zwei angeschlossenen Diskettenstationen beide Motoren anlaufen. Der Grund dafür ist der, daß die verantwortliche Signalleitung vom Computer an beiden Laufwerken gleichzeitig anliegt. Dies hat auch den Vorteil, daß bei Kopiervorgängen von einem Laufwerk auf das andere ständig beide Motoren mit Nenndrehzahl laufen, so daß viel Zeit für das Starten der Motoren gespart wird.

Nun muß daher als nächster Schritt ein einzelnes Laufwerk angesprochen werden. Dies geschieht über die Drive-Select-Leitung. Fühlt sich ein Laufwerk also angesprochen, so leuchtet die BUSY-Lampe auf und zeigt den Betrieb dieses Gerätes an.

Es folgt nun die Entscheidung, welche Daten von der Diskette gelesen werden sollen. Hierfür muß der Rechner genau angeben, auf welcher Spur diese Daten liegen. Diese Spuren, auch Tracks genannt, sind imaginäre Ringe, die konzentrisch auf der Magnetscheibe angeordnet sind. Der Schreib-/Lesekopf, der mit einem Ärmchen auf eine bestimmte Stelle der Diskette geschoben wird, gleitet somit auf der rotierenden Scheibe genau über diese Spur.

Auf diesen Spuren sind nun nach einem bestimmten System die gespeicherten Daten als winzige magnetische Punkte verteilt. Um die Verteilung der Daten innerhalb der Spur noch etwas handlicher zu machen, werden die Tracks wiederum unterteilt. diese Unterteilung nennt man Sektoren. Jede Spur trägt 9 solcher Sektoren, von denen jeder Sektor 512 Bytes wirkliche Daten enthält. Die Bezeichnung 'wirkliche Daten' deutet nun darauf hin, daß ein Sektor eigentlich noch mehr Daten enthält, die nicht unmittelbar verfügbar sind. Lassen Sie uns jedoch die Beschreibung dieser speziellen Bytes auf ein späteres Kapitel verschieben.

Innerhalb des Schreib-/Lesekopfes, der über die rotierende Magnetscheibe gleitet, liegt eine kleine Spule. Diese Spule dient als magnetischer Empfänger und kann somit die als magnetische Informationen vorhandenen Datenbits als Impulse erkennen. Dieses Prinzip erinnert an normale Tonbandgeräte, nur daß bei Disketten eine wesentlich höhere Präzision nötig ist. Schließlich ist es möglich, auf einer Fläche von ca. 30 cm (bei 3½-Zoll-Disketten) jedes einzelne von fast 3 Millionen Bits, also Ja-Nein-Informationen, genau wiederzufinden! Somit benötigt ein Byte, das ja aus 8 Bits besteht, nur eine Fläche von 0,008 mm !

Braucht der Rechner nun Daten von der Diskette, so fordert er im allgemeinen einen einzelnen Sektor von der Diskette an. Durch einen recht komplizierten Vorgang entscheidet der im Rechner eingebaute Disketten-Controller, welche der Unmengen Bits, die vom Schreib/Lesekopf kommen, diesen Sektor darstellen. Diese Datenbits werden dann herausgepickt und die erhaltenen 512 Bytes an den Computer geliefert.

Alle diese Vorgänge sind in der Praxis ein großes Problem für die Hersteller von Diskettenlaufwerken. Die Mechanik, die den Kopf positioniert, muß den gewünschten Track (etwa 0,2 mm breit) genau einstellen. Dann müssen die magnetischen Impulse, die von der rotierenden Scheibe kommen, genau als Ja oder Nein erkannt werden, wofür bei 300 Umdrehungen pro Minute nur etwa 0.5 Microsekunden pro Bit Zeit ist.

Aus all diesen Bits nun die gewünschten Daten herauszufinden ist nun die weitere Aufgabe der Elektronik. Dies wird durch sogenannte Synchronisationsbytes erreicht, die am Anfang eines jeden Sektors auf der Diskette stehen. Doch verschieben wir die Betrachtung der Datenstrukturen auf das gleichnamige Kapitel und bleiben vorerst bei der Hardware.

Ein Diskettenlaufwerk ist also, wie wir gesehen haben, eine recht komplizierte Angelegenheit. Wir wollen uns daher nur mit dem prinzipiellen Aufbau des gesamten Systems befassen, welches für die Anwendung von Disketten nötig ist.

## 4.2 Der DMA-Chip

Beginnen wir im Rechner selbst. Die Diskettenstation sendet die angeforderten Daten durch das Kabel, die nach der Aufbereitung als eine Flut von Bytes ankommen. Diese Daten müssen nun irgendwo im Speicher abgelegt werden, um sie weiterzuverwenden. Die meisten Computer gehen dabei so vor, daß die zentrale Recheneinheit (Central-Processor-Unit = CPU) die Daten in Empfang nimmt und im Speicher ablegt. Das bedeutet jedoch, daß für die Dauer der Datenübertragung nichts anderes stattfinden kann.

Der ATARI ST dagegen arbeitet anders. Den Empfang und die Verteilung der Daten im Speicher übernimmt ein spezieller Baustein, der ebenso wie die CPU einen direkten Zugriff zum Arbeitsspeicher hat. Dieser Baustein heißt DMA-Chip (DMA = Direct Memory Access). Er arbeitet, natürlich nur auf einen entsprechenden Befehl der CPU hin, völlig selbstständig, so daß die CPU während der Datenübertragung an anderen Aufgaben arbeiten kann. Außerdem kann der DMA-Chip die Datenübertragung wesentlich schneller abwickeln, als es die CPU könnte.

Durch diesen hardwaremäßigen Kunstgriff der ATARI-Konstrukteure ist die erreichbare Übertragungsgeschwindigkeit sehr hoch, was sich bei Diskettenoperationen und in noch größerem Maße bei Festplatten angenehm auswirkt.

Der DMA-Chip belegt im Speicher des ATARI ST folgende Speicherzellen:

*$FF8604* — *FDC-Access/Sector Count.* Hier wird auf das Register des DMA- oder FDC-Chips zugegriffen, welches ausgewählt wird mit

*$FF8606* — *DMA-Mode/ Status.* Die Bits 0-2 ergeben beim Lesen den Status des DMA- und FDC-Chips. Schreiben in dieses 16-Bit-Register setzt den Modus des DMA-Chips.

| | |
|---|---|
| *$FF8609* | *DMA-Speichervektor HI-Byte* |
| *$FF860B* | *MID-Byte* |
| *$FF860D* | *LO -Byte* |

Diese 3 Bytes ergeben die 24-Bit-Adresse, an die bzw. von der die Daten von der DMA übertragen werden sollen. Diese Bytes müssen unbedingt in der Reihenfolge LO,MID,HI eingetragen werden.

Der im ATARI ST verwendete DMA-Chip liegt direkt an der linken Schnittstelle des ST, an die die Festplatte angeschlossen werden kann. Der Anschluß der Diskettenlaufwerke ist nicht direkt mit ihm verbunden. Zwischen dieser Buchse und dem DMA-Chip liegt das Bauteil, welches die seriell ankommenden Daten aufbereitet bzw. die zur Diskette zu sendenden Daten seriell abschickt. Dieser Baustein ist der sogenannte Floppy-Disk-Controller, der auch die Funktionen des Laufwerks steuert. Die Programmierung dieser beiden Bausteine ist so vernetzt, daß wir sie im nächsten Abschnitt verdeutlichen wollen.

### 4.2.1 Der Disk-Controller

Dieses zugegebenermaßen große Kapitel handelt von dem Floppy-Disk-Controller WD1772 (im weiteren nur noch FDC oder Controller genannt), der im ATARI ST verwendet wird. Doch in welchem anderen Buch, als dem Floppy-Buch, hätte ein derart umfangreiches Kapitel, zumal es nur einen einzigen Baustein beschreibt, seine Berechtigung? Wir haben alle uns erhältlichen Informationen und Datenblätter über diesen Chip zusammengetragen. Diese allein reichten für eine umfassende Beschreibung natürlich nicht aus, da sich Theorie und Praxis oft voneinander unterscheiden. So war es nötig eigene Erfahrungen über den WD1772 zu sammeln, die die Richtigkeit der vorhandenen Informationen bestätigen bzw. Abweichungen von ihnen aufdecken sollten.

Das daraus entstandene Kapitel enthält für denjenigen, der sich nur einen globalen Überblick über den Controller verschaffen möchte, ein Übermaß an Informationen. Dem Anwender, der über ausreichende Programmiererfahrung verfügt, um den FDC

direkt anzusteuern zu können und sich nur aus mangelnder Kenntnis nicht an ihn heranwagt, wird hierdurch allerdings das nötige Wissen über diesen Chip vermittelt.

Für den normalen Datenaustausch zwischen Diskettenlaufwerk und ATARI ST ist es nicht erforderlich, den FDC in Eigenregie zu programmieren. Diese Aufgabe kann, durch entsprechende Aufrufe des BIOS bzw. XBIOS, dem Betriebssystem übergeben werden.

Von Seiten des Betriebssystems werden jedoch nicht alle Möglichkeiten, die der FDC bietet, unterstützt. Für den Anwender, der z.B. ein schnelles Kopierprogramm oder einen Kopierschutz entwickeln möchte, sind die fehlenden Funktionen allerdings von großer Bedeutung. Wer besondere Diskettenformate erstellen möchte, kann nicht die vorhandene Betriebssystemroutine zur Spur-Formatierung benutzen, sondern muß hier selbst aktiv werden.Um all diese Funktionen in ein Anwenderprogramm einzubinden, muß man sich der direkten Programmierung des Controllers bedienen. Dies ist jedoch nur bei genauer Kenntnis der FDC-Kommandos und deren Abläufe möglich. Diese Kenntnis erspart ferner ein unnötiges Experimentieren, um nach stundenlangem programmieren feststellen zu müssen, daß man seine Idee doch nicht verwirklichen kann.

Ein Beispiel einer solchen Idee wäre: "Wenn ich alle Spuren einer Diskette, mittels des READ-TRACK-Kommandos, in den Rechner einlese und danach alle Spuren, durch das WRITE-TRACK-Kommando, auf eine andere Diskette schreibe, so habe ich das schnellste Kopierprogramm das man sich vorstellen kann. Obendrein habe ich damit sogar die Möglichkeit, ein 'Back Up' von meinen kopiergeschützten Programmdisketten zu fertigen. Durch den READ-TRACK-Befehl werden ja alle Informationen der Spur (also auch der Kopierschutz) gelesen und durch das WRITE-TRACK-Kommando werden diese wieder vollständig geschrieben!"

Wenn Sie nun ein Programm schreiben, das nach diesem Schema arbeitet, so werden Sie feststellen, daß die Kopien schlichtweg

unbrauchbar sind. Es funktioniert noch nicht mal die Kopie einer ungeschützten Diskette.

Wenn Sie dieses Kapitel durchgearbeitet haben und danach wissen, was die Kommandos des FDC bewirken und wie sie im einzelnen ablaufen, so werden Sie erkennen, weshalb der Versuch des beschriebenen "Kopierprogramms" zum scheitern verurteilt ist. Wir sind der Überzeugung, daß die Beschreibung des Controllers umfassend genug ist, um "Ideen" dieser Art zu verhindern.

Doch nun endlich zur Beschreibung des WD1772. Dieser von WESTERN DIGITAL entwickelte Chip vereint in sich alle Funktionen, die zur Steuerung eines 5¼-Zoll-Laufwerks notwendig sind. Natürlich lassen sich auch 3½-Zoll-Laufwerke, wie ja von ATARI demonstriert, problemlos mit diesem Chip steuern. Diese Möglichkeit ist jedoch nicht besonderen Fähigkeiten des WD1772 zu verdanken, sondern dem Entwickler der 3½-Zoll-Laufwerke, der Firma SONY. Dort kam man zu dem Schluß, daß es einer schnellen Markteinführung zuträglich wäre, wenn man den 3½-Zoll-Laufwerken, eine zu den 5¼-Zoll-Laufwerken kompatible Schnittstelle spendieren würde. Aus der Sicht des ATARI ST Besitzers heißt das natürlich, daß er auch 5¼-Zoll-Laufwerke anschließen kann.

Doch Vorsicht bei älteren Laufwerken, die Sie vielleicht noch besitzen oder ihnen als Industrie-Restposten günstig angeboten werden. Wenn Sie ein solches Modell als Fremdlaufwerk anschließen möchten, kann es aus folgendem Grund Probleme geben:

Der WD1772, dessen Standardversion WD1770, mit den älteren FDC-Serien WD179x und WD279x softwarekompatibel ist, verfügt über kürzere "Stepping Rates", das sind die Zeiten, die der Controller dem Laufwerk zur Verfügung stellt, um den Schreib/Lese-Kopf eine Spur nach innen oder außen zu bewegen.

Beim WD1772 liegen die vier programmierbaren Zeiten bei 2, 3, 5 und 6 ms, während beim WD1770 die Zeiten 6, 12, 20 und 30

ms betragen. Dies bedeutet, daß die Laufwerke in der Lage sein müssen, einen Spurwechsel in max. 6ms zu vollziehen. Schauen Sie in das Datenblatt des betreffenden Laufwerks. Dort finden Sie unter "TRACK TO TRACK" die Zeit die das Laufwerk benötigt.

Doch das nur als Hinweis nebenher. Wenden wir uns nun wieder dem FDC zu und beginnen mit einer kurzen Zusammenfassung der einzelnen Leistungsmerkmale dieses Chips.

Die "Features" des WD1772 sind:

- 28 Pin Dual-in-line Gehäuse
- Einfache 5V Stromversorgung
- eingebauter digitaler Datenseparator
- eingebaute Schreib-Vorkompensation
- einfache und doppelte Schreibdichte
- eingebaute Motorkontrolle
- Sektorlänge 128, 256, 512 oder 1024 Byte
- schnelle "Stepping Rates" (2, 3, 5 und 6ms)

Wie gesagt, dies sind nur die Besonderheiten im Überblick. Zwei dieser Punkte möchten wir direkt erläutern, die übrigen werden in den folgenden Kapiteln, im Zusammenhang mit den einzelnen Funktionen des FDC, ausführlich erklärt.

1. Das der WD1772 in einem 28-Pin-Gehäuse untergebracht ist, ist wohl nur für die Entwicklung eines Systems, in welchem ein FDC benötigt wird, interessant. Da bei einem Platinen-Layout ein 28-poliger Chip weniger Aufwand als z.B. ein 40-poliger Chip verursacht, wird einem System-Entwickler hierdurch eine Entscheidungshilfe (sofern die Leistungsmerkmale überzeugen) geliefert.

2. Auf die Möglichkeit, den WD 1772 in einfacher oder doppelter Schreibdichte (single Density/double Density) betreiben zu können, wird im Verlauf der Controller-Beschreibung nicht weiter eingegangen. Der Grund dafür ist einleuchtend. Der FDC wird im

ATARI ST mit doppelter Schreibdichte betrieben. Um den Controller mit einfacher Schreibdichte zu betreiben, müßte der Rechner geöffnet und die Beschaltung des FDC geändert werden. Der Erfolg dieses Unternehmens wäre, daß man auf einer Diskette nur noch 50% der üblichen Speicherkapazität erreicht. Was auf den ersten Blick als unsinnig erscheint (wer verzichtet schon freiwillig auf die Hälfte des Speichervermögens), kann in der Praxis den Vorteil bringen, ein zum Computer XYZ kompatibles Diskettenformat zu erzeugen. Doch solche speziellen Anwendungsfälle sind sicherlich nicht von allgemeinem Interesse. Da die Thematik des FDC auch so schon komplex genug ist, möchten wir Sie nicht mit den Möglichkeiten des FDC belasten, die im ATARI ST wahrscheinlich nie angewendet werden.

#### 4.2.1.1 Anschlußbelegung

Nach den globalen Vorbetrachtungen zum WD 1772 starten wir die ausführliche Beschreibung - wie sollte es anders sein - mit der Anschlußbelegung des FDC.

WD 1772

| Signal | Pin | Pin | Signal |
|---|---|---|---|
| $\overline{CS}$ | 1 | 28 | INTRQ |
| R/$\overline{W}$ | 2 | 27 | DRQ |
| A0 | 3 | 26 | $\overline{DDEN}$ |
| A1 | 4 | 25 | $\overline{WPRT}$ |
| DAL 0 | 5 | 24 | $\overline{IP}$ |
| DAL 1 | 6 | 23 | $\overline{TR00}$ |
| DAL 2 | 7 | 22 | WD |
| DAL 3 | 8 | 21 | WG |
| DAL 4 | 9 | 20 | MO |
| DAL 5 | 10 | 19 | $\overline{RD}$ |
| DAL 6 | 11 | 18 | CLK |
| DAL 7 | 12 | 17 | DIRC |
| $\overline{MR}$ | 13 | 16 | STEP |
| GND | 14 | 15 | $V_{CC}$ |

**PIN 1 CS (CHIP SELECT)**

Ein LOW an diesem Eingang selektiert den Chip und ermöglicht dadurch den Zugriff auf seine Register. Den CHIP-SELECT-Anschluß finden Sie an allen Peripheriebausteinen (wozu natürlich auch die Speicher-IC's zählen). Da diese allesamt an dem Datenbus des Prozessors angeschlossen sind, würde es dort zu einem wilden Durcheinander kommen wenn alle gleichzeitig an dem Datenverkehr teilnähmen. Deshalb wird über den CHIP SELECT Eingang nur der Baustein eingeschaltet, mit dem ein Datenaustausch stattfinden soll. Im ATARI ST wird der FDC über den DMA-Controller selektiert.

**PIN 2 R/W (READ/WRITE)**

Der Pegel an diesem Eingang steuert die Datenrichtung. Bei einem HIGH wird der Inhalt des selektierten Registers auf DAL0-DAL7 ausgegeben, während bei einem LOW die Daten auf DAL0-DAL7 in das selektierte Register übernommen werden.

**PIN 3,4 A0,A1 (ADDRESS 0,1)**

Mit diesen beiden Eingängen werden die Register des FDC selektiert. Der WD1772 besitzt 5, vom Computer- System, adressierbare Register. Da mit zwei Adressen-Leitungen jedoch nur 4 Register ausgewählt können, wurde eine Adresse (A0=0 und A1=0) mit zwei Registern belegt. Zur Unterscheidung dieser Register wird der Pegel an dem R/W-PIN ausgewertet.

| CS | A1 | A0 | R/W = 1 | R/W = 0 |
|---|---|---|---|---|
| 0 | 0 | 0 | Status Reg. | Command Reg. |
| 0 | 0 | 1 | Track Reg. | Track Reg. |
| 0 | 1 | 0 | Sector Reg. | Sector Reg. |
| 0 | 1 | 1 | Data Reg. | Data Reg. |

Daraus resultiert, daß das Command-Reg. nicht gelesen, bzw. das Status-Reg. nicht beschrieben werden kann. Diese Anschlüsse sind nicht direkt mit dem Adressbus des Prozessors verbunden, sondern sind an den DMA-Controller angeschlossen. Die Register des FDC werden über ein Steuer-Register im DMA-Controller selektiert.

## PIN 5-12 DAL0-DAL7 (DATA-ACCESS-LINE 0-7)

Diese 8 Leitungen bilden den bidirektionalen Datenbus. Über diesen Bus werden die Daten zwischen Computersystem und den FDC-Registern übertragen. Diese Leitungen sind, genau wie die Adressen- leitungen, mit dem DMA-Controller verbunden. Über dessen Datenregister wird indirekt auf jenes Register des FDC zugegriffen, welches über das Steuerregister des DMA-Controllers selektiert wurde.

## PIN 13 MR (MASTER RESET)

Da nach dem Anlegen der Versorgungspannung an den FDC der Inhalt seiner Register rein zufällig ist, müssen diese in einen definierten Grundzustand versetzt werden. Dies erreicht man durch einen LOW- Impuls (von mindestens 50us) an diesem Eingang. Dies geschieht üblicherweise nach dem Einschalten des ATARI ST. Durch den RESET-Befehl des 68000 Prozessors besteht natürlich jederzeit die Möglichkeit, den FDC zurückzusetzen. Hierbei sollte jedoch beachtet werden, daß auch alle anderen Bau-

steine, die mit der gemeinsamen Resetleitung verbunden sind, zurückgesetzt werden und eventuell mit bestimmten Startwerten initialisiert werden müssen.

**PIN 14 GND (GROUND)**

Masseanschluß

**PIN 15 Vcc (POWER SUPPLY)**

Eingang der +5V Stromversorgung

**PIN 16 STEP (STEP)**

Über diesen Ausgang wird dem Laufwerk für jeden Schritt, den der Schreib/Lese-Kopf bewegt werden soll, ein Impuls geliefert.

**PIN 17 DIRC (DIRECTION)**

Über den Pegel an diesem Ausgang zeigt der FDC dem angeschlossenen Laufwerk an, in welche Richtung es den Schreib/Lese-Kopf bei Eintreffen eines STEP- Impulses bewegen soll. Liegt dieser Anschluß auf HIGH-Pegel, so bewirkt ein STEP-Impuls einen Schritt zur Diskettenmitte, während durch einen LOW-Pegel der STEP-Impuls einen Schritt nach außen veranlaßt.

**PIN 18 CLK (CLOCK)**

Durch die Übergabe eines Befehls gestartet, laufen im FDC, ähnlich wie in einem Mikroprozessor, Mikroprogramme ab. Aus diesem Grunde ist der FDC, genau wie ein Mikroprozessor, auf einen Takt ange-

wiesen, der dessen Ausführung steuert. Auch benötigt er diesen Takt für das Timing des seriellen Datenstroms. Der Takt wird nicht im FDC selbst erzeugt, sondern diesem von außen, über den CLK-Eingang, zugeführt. Die Taktfrequenz liegt bei 8 MHz.

**PIN 19 RD (READ DATA)**

Das Signal, das der Schreib/Lese-Kopf des Laufwerks liefert, wird an diesen Eingang des FDC angelegt. Im Datenseparator des FDC werden Takt- und Daten-Impulse, die beide im Signal enthalten sind, voneinander getrennt.

**PIN 20 MO (MOTOR ON)**

Dieser Ausgang wird zur Motorsteuerung benutzt. Durch einen HIGH-Pegel werden die Antriebsmotoren der Laufwerke, bei Schreib-, Lese- und Such-Operationen, vom FDC automatisch gestartet.

**PIN 21 WG (WRITE GATE)**

Die Daten-Impulse, die der FDC zum Laufwerk überträgt, erreichen dort nicht direkt den Schreib/Lese-Kopf, sondern zunächst einen Schreibverstärker. Werden keine Daten vom FDC übertragen, so ist der Eingang dieses Verstärkers offen. Verstärker mit offenen Eingänge haben aber die unangenehme Eigenschaft, für Fremdspannungen, die z.B in das Verbindungskabel eingestreut werden, empfänglich zu sein. Um zu verhindern, daß solche unerwünschten Spannungen den Schreib/Lese-Kopf erreichen und dadurch auf der Diskette Daten zerstört werden, besitzen die Laufwerke eine Schaltung, die den Schreibverstärker verriegelt. Bei Schreibo-

perationen, also nur wenn Daten geschrieben werden sollen, wird WRITE GATE vom FDC auf HIGH-Pegel gelegt. Dadurch wird die Verriegelung des Schreibverstärkers aufgehoben und die, über WRITE DATA eintreffenden Datenimpulse, können von diesem verarbeitet werden.

**PIN 22 WD (WRITE DATA)**

Über diesen Ausgang wird die zu schreibende Information, bestehend aus Daten- und Taktimpulsen, zum Laufwerk übertragen.

**PIN 23 TR00 (TRACK 00)**

Die Laufwerke verfügen über eine Lichtschranke, die durch den Schlitten, auf dem sich der Schreib/Lese-Kopf befindet, unterbrochen wird, sobald sich der Kopf über der Spur Null befindet. In diesem Fall wird der TR00-Eingang des FDC vom Laufwerk auf LOW gelegt.

**PIN 24 IP (INDEX PULSE)**

Das Laufwerk liefert bei jeder Umdrehung des Antriebmotors über diesen Anschluß einen Impuls, der von dem Controller bei seinen Operationen ausgewertet wird. So erkennt er z.B. beim Lesen oder Schreiben einer Spur hierüber deren Anfang (Der Index-Impuls ist praktisch die Antwort auf die Frage: Wo beginnt ein Kreis ?).

Auch kann er durch Zählen der Index-Impulse die Hochlaufzeit des Motors berücksichtigen und auf diese Weise etwas warten, bis er seine Solldrehzahl erreicht hat.

Der Index-Impuls wird bei den 3½-Zoll Laufwerken unabhängig von der Diskette erzeugt. Es wird hier also kein Indexloch in der Diskette (wie beim 5¼-Zoll Format) durch eine Lichtschranke abgetastet.

**PIN 25 WPRT (WRITE PROTECT)**

Soll der FDC eine Schreib-Operation ausführen, so wird zunächst dieser Eingang vom FDC abgefragt. Falls dieser Eingang vom Laufwerk auf LOW gelegt wurde (schreibgeschützte Diskette), so bricht der Controller die Schreib-Operation ab.

**PIN 26 DDEN (DOUBLE DENSITY ENABLE)**

Der Pegel an diesem Eingang bestimmt das Aufzeich-nungsformat, mit dem der FDC arbeitet. Durch ein LOW wird der Controller im DOUBLE DENSITY Modus (doppelte Aufzeichnungsdichte) betrieben, während durch ein HIGH der SINGLE DENSITY Modus eingeschaltet ist. Im ATARI ST wird der WD1772 grundsätzlich im Double Density Modus betrieben, da DDEN mit Masse beschaltet ist und somit auf LOW liegt.

**PIN 27 DRQ (DATA REQUEST)**

Dieser Ausgang, dessen Zustand auch durch ein Bit im Status-Register angezeigt wird, hat, wenn er vom FDC auf HIGH gelegt wird, folgende Bedeutung:

a. Bei einer Leseoperation befindet sich ein Byte im Datenregister, das nun ausgelesen werden muß (Datenregister voll).

b. Bei einer Schreiboperation ist das Datenregister leer und muß nun mit dem nächsten zu schreibenden Byte geladen werden.

Ein Lesen oder Schreiben des Datenregisters setzt den DRQ-Ausgang und das DRQ-Statusbit wieder zurück.

Die DMA-Fähigkeit des WD1772 beruht auf dem Vorhan-densein dieses Ausgangs. Im ATARI ST ist dieser mit dem DMA-Controller verbunden. Während man sonst, bei Lese- und Schreib-Operationen, das DRQ-Statusbit abfragen muß, um zu erkennen wann ein Datentransfer stattfindet, wird diese Aufgabe vom DMA-Controller, durch den DRQ-Ausgang gesteuert, selbstständig erledigt.

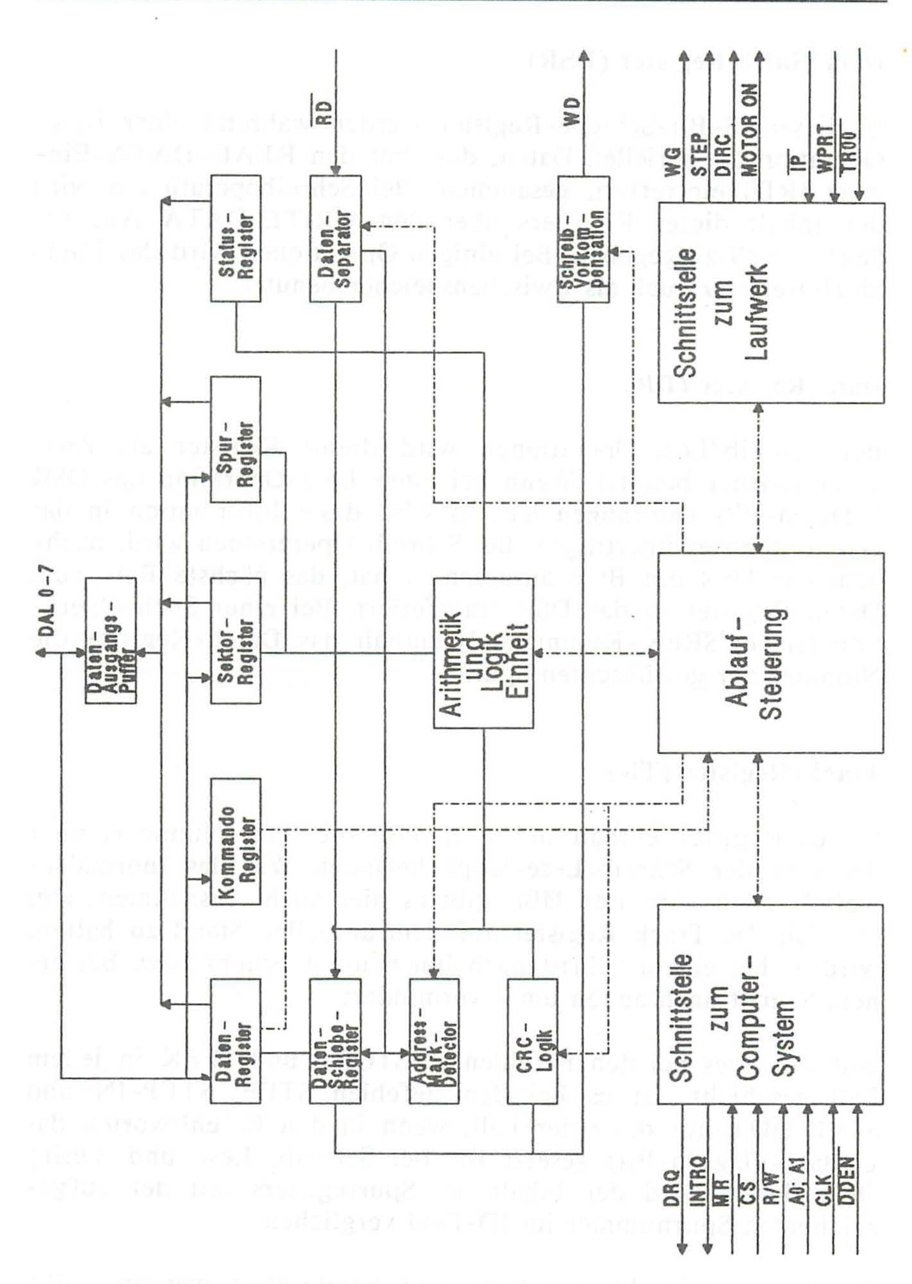
**PIN 28 INTRQ (INTERRUPT REQUEST)**

Nach jedem beendeten Kommando wird dieser Ausgang vom FDC auf HIGH gelegt. Durch ein Lesen des Status-Registers wird der Ausgang wieder zurückgesetzt. Dieser Anschluß ist mit dem I/O-Port (Bit 5) des MFP 68901 verbunden. Um zu erkennen wann der FDC sein Kommando beendet hat, wird dieses Port-Bit in einer Schleife abgefragt. Zu beachten ist dabei, daß dieses Bit invertiert ist. Das Kommando ist also beendet, wenn das Port-Bit gelöscht ist.

Es besteht die Möglichkeit den MFP so zu programmieren, daß er, bei einem HIGH des INTRQ-Ausgangs, einen Interrupt auslöst. Dadurch spart man die ständige Abfrage des Port-Bits und kann stattdessen schon andere Aufgaben erledigen. Die Interruptsteuerung wird vom Betriebssystem nicht benutzt.

### 4.2.1.2 Organisation

Um die Programmierung des FDC, die später ausführlich erklärt wird, besser zu verstehen, wäre es günstig, sich zunächst einmal ein Blockschaltbild des WD1772 anzuschauen und danach die einzelnen Funktionsblöcke zu erklären.



DAL 0-7
Daten-Ausgangs-Puffer
Status-Register
Spur-Register
Sektor-Register
Kommando Register
Daten-Register
Daten-Schiebe-Register
Address-Mark-Detector
CRC-Logik
Daten Separator
RD
Arithmetik und Logik Einheit
Schreib-Vorkom-pensation
WD
WG
STEP
DIRC
MOTOR ON
IP
WPRT
TR00
Schnittstelle zum Laufwerk
Ablauf-Steuerung
Schnittstelle zum Computer-System
DRQ
INTRQ
MR
CS
R/W
A0, A1
CLK
DDEN

**Data-Shift-Register (DSR)**

In diesem 8-Bit-Schiebe-Register werden während einer Lese-Operation die seriellen Daten, die über den READ-DATA-Eingang (RD) eintreffen, gesammelt. Bei Schreiboperationen wird der Inhalt dieses Registers über den WRITE-DATA Ausgang (WD) seriell ausgegeben. Bei einigen Operationen wird das Data-Shift-Register auch als Zwischenspeicher benutzt.

**Data-Register (DR)**

Bei Schreib/Lese-Operationen wird dieses Register als Zwischenspeicher benutzt. Wenn bei einer Lese-Operation das DSR 8 Daten-Bits empfangen hat, so wird diese Information in das Daten-Register übertragen. Bei Schreib-Operationen wird, nachdem das DSR ein Byte ausgesendet hat, das nächste Byte vom Daten-Register in das DSR transferiert. Bei einer Such-Operation (siehe 'SEEK-Kommando') enthält das Daten-Register die Nummer der gewünschten Spur.

**Track-Register (TR)**

Dieses Register enthält normalerweise die Spur-Nummer, über der sich der Schreib/Lese-Kopf befindet. Wie das "normalerweise" schon vermuten läßt, gibt es hier auch Ausnahmen. Um nämlich das Track-Register auf dem aktuellen Stand zu halten, wird es bei einem Schritt nach innen um 1 erhöht bzw. bei einem Schritt nach außen um 1 vermindert.

Während dies bei den Befehlen RESTORE und SEEK in jedem Fall geschieht, ist es bei den Befehlen STEP, STEP-IN und STEP-OUT nur dann der Fall, wenn in den Befehlsworten das Update-Flag (u-Bit) gesetzt ist. Bei Schreib, Lese und Verify Operationen wird der Inhalt des Spurregisters mit der aufgezeichneten Spurnummer im ID-Feld verglichen.

Das Spurregister kann gelesen und beschrieben werden, sollte während einer Operation jedoch nicht geladen werden.

**Sector-Register (SR)**

Bei Schreib und Leseoperationen enthält dieses Register die Nummer des gewünschten Sektors, welche mit der im ID-Feld aufgezeichneten Sektornummer verglichen wird. Nach einem READ ADDRESS Befehl befindet sich die Spurnummer aus dem ID-Feld in diesem Register.

Das Sektor-Register kann gelesen und beschrieben werden. Während einer Operation sollte das Spurregister jedoch nicht geladen werden.

**Command-Register (CR)**

Dieses Register enthält das gerade in der Ausführung befindliche Kommando. Es kann nur beschrieben werden, da ein Lesen automatisch das Status-Register selektiert. Das Kommando-Register sollte während der Ausführung einer Operation nicht geladen werden, außer wenn es sich bei dem neuen Kommando um den FORCE-INTERRUPT-Befehl handelt.

**Status-Register (STR)**

Die in diesem Register befindliche Information gibt Aufschluß über Zustand des FDC bzw. des Laufwerks. Die einzelnen Bits dieses Registers werden teilweise abhängig vom bearbeiteten Befehl gesetzt. Das Status-Register kann nur gelesen werden, wobei das gelesene Byte folgende Bedeutung(en) hat:

**Bit 7 MOTOR ON**

Dieses Bit reflektiert den Zustand des MOTOR-ON-Aus-gangs. Es ist nach einem Kommando, also wenn das Busy-Bit schon gelöscht ist, noch für ca. 1 bis 2s gesetzt.

**Bit 6 WRITE PROTECT**

Dieses Bit zeigt nach Schreib-Operationen an, ob sich im Laufwerk eine schreibgeschützte Diskette befindet. Ist es gesetzt, so bedeutet das gleichzeitig, daß die gewünschte Schreib-Operation nicht ausgeführt wurde. Das WPRT-Bit wird ebenfalls (im Falle einer schreibgeschützten Diskette) nach einem Typ-1-Kommando gesetzt.

Zurückgesetzt wird dieses Bit, wenn eine Operation mit einer, nicht schreibgeschützten, Diskette stattgefunden hat.

**Bit 5 SPIN UP/RECORD TYPE**

*SPIN UP:* Bei Typ-1-Kommandos wird dieses Bit nach Abschluß der Spin-Up-Sequenz gesetzt. Damit soll angezeigt werden, daß der Laufwerksmotor seine Nenndrehzahl (wahrscheinlich) erreicht hat.

*RECORD TYPE:* Nach einem READ-SECTOR-Befehl läßt sich hieran erkennen, ob das Daten-Feld mit einem 'normalen' oder 'gelöschtem' Data-Mark beginnt.

Bit 5 = 0, 'normales' Data-Mark ($FB)
Bit 5 = 1, 'gelöschtes' Data-Mark ($F8)

**Bit 4 RECORD NOT FOUND (RNF)**

Wird kein korrektes ID-Feld gefunden, so wird dieses Bit gesetzt. Dies kann nach einem READ-SECTOR-, WRITE-SECTOR- oder READ-ADDRESS-Kommando der Fall sein.

Nach einen READ SECTOR Befehl kann das RNF-Bit aber auch trotz eines korrekten ID-Feldes gesetzt

sein. Dies ist dann der Fall, wenn innerhalb der 43 Bytes, die dem letzten CRC-Byte des ID-Feldes folgen, kein Data-Mark gefunden wurde.

**Bit 3 CRC ERROR**

Dieses Bit wird gesetzt, wenn der Inhalt des CRC-Feldes (im Daten- oder ID-Feld) nicht mit dem Inhalt des CRC-Registers übereinstimmt.

**Bit 2 LOST DATA / TRACK 00**

*LOST DATA:* Wird bei Typ-2- und Typ-3-Kommandos nicht innerhalb der erforderlichen Zeit auf eine Datenanforderung (angezeigt durch DRQ-Ausgang bzw. DRQ-Statusbit) reagiert, so wird dieses Bit gesetzt.

*TRACK 00:* Bei Typ-1-Kommandos ist dieses Bit gesetzt, wenn sich der Schreib/Lese-Kopf über der Spur Null befindet.

**Bit 1 DATA REQUEST / INDEX**

*DATA REQUEST:* Bei Typ-2 und Typ-3 Kommandos wird dieses Bit gesetzt wenn Daten bereitstehen bzw. benötigt werden. Es wird durch Lesen oder Schreiben des Daten-Registers zurückgesetzt.

*INDEX:* Bei Typ-1 Kommandos ist dieses Bit während eines eintreffenden Index-Impulses gesetzt.

**Bit 0** **BUSY**

Während der Ausführung eines Kommandos ist dieses Bit gesetzt.

### CRC-Logic

Um Lesefehler zu vermeiden, muß man sich eines Verfahrens bedienen, das eine hohe Datensicherheit gewährleistet. Das hier angewandte Verfahren funktioniert so, daß aus den geschriebenen Daten, nach einem bestimmten Algorithmus eine 16-Bit Prüfsumme gebildet wird, welche zusätzlich, im Anschluß an die Daten, auf die Diskette geschrieben wird.

Werden nun diese Daten wieder gelesen, so wird nach dem gleichen Algorithmus erneut die Prüfsumme gebildet. Stimmt diese mit der aufgezeichneten überein, so hat man eine fast 100 prozentige Sicherheit, daß die Daten richtig gelesen wurden. Durch den relativ komplizierten Algorithmus ist es nämlich sehr unwahrscheinlich, daß sich trotz Lesefehler die gleiche Prüfsumme ergibt.

Diese 16-Bit Prüfsumme wird "Cyclic Redundancy Check (CRC)" genannt. Für die Erzeugung und Kontrolle der Prüfsumme ist die CRC-Logik zuständig. Die Berechnung erfolgt aus Geschwindigkeitsgründen durch eine Hardware, die die Summe nach dem Polynom : CRC(x)= x16 + x12 + x5 + 1 berechnet.

### ARITHMETIC/LOGIC UNIT (ALU)

Die ALU wird einerseits zur Registermodifikation (erhöhen, vermindern), andererseits für Vergleiche zwischen Register und den auf der Diskette enthaltenen Informationen in den ID-Feldern benutzt.

**ADDRESS-MARK-DETECTOR**

Das wahrscheinlich wichtigste Teil im FDC ist dieser Detektor. Wie der Name schon verrät, besitzt dieses Teil die Fähigkeit, ein Address Mark zu erkennen. Eine solche Markierung kennzeichnet den Anfang eines ID-Feldes (Index- Address-Mark) bzw. den Anfang eines Datenfeldes (Data- Address-Mark).

Doch wozu braucht man einen speziellen Detektor? Nehmen wir als Beispiel einmal den Wert '$FE', den der Controller als ein Index-Address-Mark interpretiert. Auch ohne besonderen Detektor ist es keine Schwierigkeit dieses '$FE' zu finden.

Das Problem wird allerdings dann deutlich, wenn man bedenkt, daß dieser Wert aber auch durchaus in einem Daten-Feld vorkommen kann. An dieser Stelle darf er jedoch nicht als Index-Address-Mark gewertet werden. Wie ist es also möglich ein '$FE' von einem anderen '$FE' zu unterscheiden? Nun, genau diese Funktion erfüllt der Address-Mark-Detector.

Wie zuvor schon erwähnt, besteht die geschriebene Information nicht nur aus Daten-Impulsen. Es sind gleichermaßen auch Takt-Impulse darin enthalten. Diese werden bei Lese-Operationen vom Datenseparator aus dem Signal herausgefiltert und dem Address-Mark-Detector zugeführt.

Bei einem WRITE-TRACK-Kommando werden Werte, die größer als als '$F4' sind, in besonderer Weise behandelt. Gemeinsam gilt für diese Werte jedoch, daß sie ohne Takt-Impulse geschrieben werden. Die so geschriebenen Werte bestehen also nur aus Daten-Impulsen.

Wird nun diese Information gelesen, so werden keine Takt-Impulse vom Datenseparator geliefert, da ja keine in dem Signal vorhanden sind. Erst durch das Fehlen der Takt-Impulse wird der Address-Mark-Detector aktiviert. Er kann also ein Address-Mark nur dann erkennen, wenn es ohne Takt-Impulse geschrieben wurde.

Es existiert aber noch ein weiteres Problem, das ihnen wahrscheinlich noch gar nicht bewußt wurde, weil bisher nur von "vollständigen" Daten-Bytes die Rede war. Diese werden allerdings seriell aufgezeichnet, wobei jedoch der Anfang eines Bytes in keiner Weise gekennzeichnet ist.

Wenn nun bei einer Lese-Operation 8 Bit im DSR gesammelt wurden, so kann man nicht davon ausgehen, daß diese tatsächlich zu einem einzigen Daten-Byte gehören. Es könnten genausogut jeweils 4 Bit, aus 2 verschiedenen Daten-Bytes, darin enthalten sein.

Der Address-Mark-Detector würde ein Address-Mark nur dann erkennen, wenn das Sammeln der Daten-Bits zufällig Byte-Synchron verläuft. Daß hier nichts dem Zufall überlassen werden darf, ist leicht einzusehen. Man muß also eine Möglichkeit haben, den Anfang eines Bytes zu erkennen. Dies geschieht durch Synchronisations-Bytes. Diese (jeweils 3) werden beim Formatieren einer Spur vor jedes Address-Mark geschrieben. Die 'SYNC-Bytes' enthalten, genau wie die Address-Marks, keine Takt-Impulse und aktivieren dadurch den Address-Mark-Detector.

Der Controller, der natürlich ständig über den Zustand des Address-Mark-Detectors unterrichtet ist, liest nun solange die seriellen Datenbits, bis der Inhalt im DSR einem SYNC-Byte, dessen Wert er ja kennt, entspricht. Ab diesem Punkt muß zwangsläufig das nächste eintreffende Bit, das erste Bit des folgenden Bytes sein.

Eines ist aber noch zu bemerken: Während der Detektor eingeschaltet ist, können die gelesenen Bytes verfälscht werden ( siehe 'READ-TRACK-Kommando'). Weshalb und wann? Wenn der Detektor durch die fehlenden Takt-Impulse aktiviert wird, so geht der FDC davon aus, daß SYNC-Bytes mit einem anschließendem Address-Mark folgen. In der Synchronisationsphase, also während der Rekonstruktion eines SYNC-Bytes, werden einige der gelesenen Daten-Bits verworfen. Sollte sich der Detektor 'aus Versehen' in einem Daten-Feld auf die Suche

nach einem Address-Mark begeben, so werden die Daten bis zum Erkennen des 'falschen Alarms' natürlich verändert.

Da der Address-Mark-Detector, empfindlich wie er nun mal ist (und auch sein muß), dazu neigt übersensibel auf fehlende Takt-Impulse zu reagieren, wird er vorsorglich (während ID- bzw. Daten-Felder gelesen werden) ausgeschaltet.

## DATA SEPARATOR

Der Datenseparator ist eigentlich schon bei den Ausführungen zum Address-Mark-Detector beschrieben worden. Deshalb sei hier nur kurz erwähnt, daß er die Aufgabe hat, aus dem gelesenen Signal die Takt-Impulse zu entfernen, die, sozusagen als Abfallprodukt, zur Steuerung des Address-Mark- Detectors benutzt werden.

### Die Schnittstelle zum Computer-System

Das Prozessor-Interface besteht aus den 8 bidirektionalen Daten-Leitungen (DAL0-DAL7), den beiden Adressen-Leitungen (A0,A1), der Datenanforderung (DRQ), der Interruptanforderung (INTRQ), dem Chip Select (CS), der Schreib/Lese-Leitung (R/W), dem Clock-Eingang (CLK) und dem Master-Reset-Eingang (MR). Über diese Anschlüsse wird der Austausch von Daten und Steuersignalen zwischen Prozessor und FDC abgewickelt.

### Die Schnittstelle zum Laufwerk

Die Informationen die der Controller vom Laufwerk erhält sind:

a. ob eine schreibgeschützte Diskette eingelegt ist (WPRT=1)

b. ob sich der Schreib/Lese-Kopf über der Spur 0 befindet (TR00=0)

c. ob die Diskette eine vollständige Umdrehung beendet hat (IP=0)

und

d. ob die seriellen Daten die gelesen wurden (RD)

Die Signale die der Controller zum Laufwerk überträgt sind:

a. einschalten des Laufwerksmotor (MOTOR ON=1)

b. ausführen eines Schrittes des Schreib/Lese-Kopfes (STEP=1)

c. die Richtung des Schrittes (DIRC=0 oder 1)

d. einschalten der Schreiblogik (WG=1)

und

e. die seriellen Daten die geschrieben werden sollen (WD)

**Die Ablaufsteuerung**

Wird ein Kommando vom FDC ausgeführt, so müssen natürlich die einzelnen Funktionsteile in einer bestimmten Reihenfolge ein- bzw. ausgeschaltet werden. Ferner werden auch Daten zwischen den einzelnen Registern des FDC transferiert, Berechnungen ausgeführt, Eingangsleitungen abgefragt und die Zustände der Ausgangsleitungen geändert. Der gesamte zeitliche Ablauf, der ja vom übergebenen Kommando abhängig ist, wird von der Ablaufsteuerung ausgeführt bzw. überwacht.

**Lese-Operationen**

Lese-Operationen finden im allgemeinen nur durch das READ-SECTOR-Kommando statt. Die anderen Befehle, durch welche auch Daten von der Diskette gelesen werden können, dienen nur Diagnosezwecken und haben für den normalen Betrieb keine Bedeutung.

Die Sektorlängen können 128, 256, 512 oder 1024 Byte betragen. Die Sektor-Länge wird beim Formatieren durch das "Längen-Feld" (das vierte Byte im ID-Feld) bestimmt. Soll ein Sektor gelesen werden, so erkennt der Controller anhand des Längen-Feldes, welche Anzahl von Daten-Bytes, ab dem DATA-ADDRESS-MARK gelesen werden müssen. Voraussetzung für ein fehlerfreies Lesen der Datenbytes ist, daß während dieser Zeit der ADDRESS-MARK-DETECTOR ausgeschaltet wird, da durch ihn Lesefehler verursacht werden können.

**Schreib-Operationen**

Bevor auf die Diskette aufgrund eines Schreib-Befehls geschrieben werden kann, muß der WRITE GATE-Ausgang vom FDC aktiviert werden.

Als Vorsichtsmaßnahme gegen unbeabsichtigtes Schreiben geschieht dies jedoch erst nachdem als Reaktion auf den vom FDC gesetzten DRQ-Ausgang das Datenregister geladen wird. Sollte dies nicht geschehen, so wird die Befehlsausführung beendet, INTRQ gesetzt, das LOST-DATA-Statusbit gesetzt und das BUSY-Statusbit gelöscht.

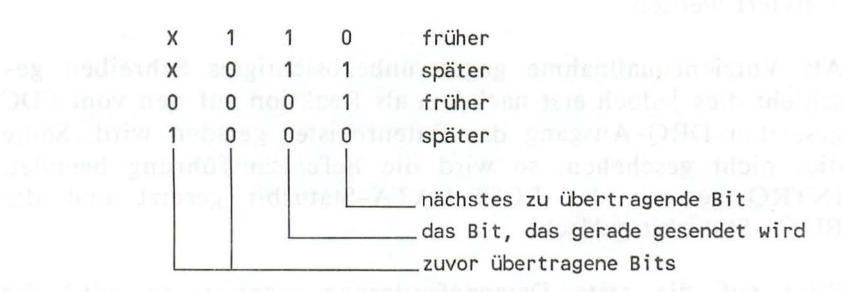
Wird auf die erste Datenanforderung reagiert, so wird das Schreib-Kommando ausgeführt. Sollte dem FDC, auf eine folgende Datenanforderung, kein weiteres Daten-Byte übertragen werden, so wird der Befehl nicht abgebrochen, sondern stattdessen ein "Null-Byte" geschrieben. Auch in diesem Fall ist nach Beendigung des Befehls das LOST-DATA- Bit gesetzt. Würden bei einem WRITE-SECTOR-Kommando also nur 112 Byte

übertragen, so würde der Controller die restlichen 400 Byte (bei einer Sektorgröße von 512 Byte) mit Nullen füllen.

Man sollte es unterlassen, diesen Sachverhalt dazu auszunutzen, einen Sektor teilweise zu löschen. Da das LOST-DATA-Bit in jedem Fall gesetzt ist, läßt sich nicht erkennen, ob nicht vielleicht bei der Übertragung der ersten Bytes ein Fehler auftrat.

Das Schreiben wird generell verhindert wenn der WRITE-PROTECT-Eingang auf LOW liegt. In diesem Fall wird jeglicher Schreib-Befehl sofort abgebrochen, INTRQ auf HIGH gelegt, das WRITE PROTECT-Statusbit gesetzt und das BUSY-Statusbit gelöscht.

Um bei einer höheren Schreibdichte, wie sie auf den inneren Spuren existiert, die Datensicherheit zu erhöhen, gibt es die Möglichkeit eine Schreib-Vorkompensation einzuschalten. Ist in den Schreibkommandos das Bit für die Schreib-Vorkompensation gelöscht, so wird der Datenstrom über WRITE- DATA, abhängig vom zu schreibenden Bitmuster, 125 ns früher oder später ausgesendet. Die folgende Tabelle zeigt, wann welcher Fall gegeben ist:




Die Schreib-Vorkompensation wird bei 5¼-Zoll-Disketten normalerweise auf den inneren Spuren, auf denen ja eine höhere Datendichte herrscht, eingeschaltet. Bei dem 3½-Zoll- Format,

auf dem die Datendichte der äußeren Spuren schon die Datendichte der mittleren Spuren des 5¼-Zoll-Formats erreicht, wird die Vorkompensation im allgemeinen ständig eingeschaltet.

### 4.2.1.3 Kommando-Beschreibung

Nachdem nun der interne Aufbau des FDC und auch einige grundlegende Abläufe bei Schreib- und Leseoperationen erklärt wurden, kommen wir zu den Kommandos.

Der WD1772 kennt 11 verschiedene Befehle (Kommandos), die in vier Gruppen bzw. Typen unterteilt sind. Die folgende Tabelle zeigt diese im Überblick.

| | | Bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Typ | Kommando | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| I | Restore | 0 | 0 | 0 | 0 | h | V | r1 | r0 |
| I | Seek | 0 | 0 | 0 | 1 | h | V | r1 | r0 |
| I | Step | 0 | 0 | 1 | u | h | V | r1 | r0 |
| I | Step-in | 0 | 1 | 0 | u | h | V | r1 | r0 |
| I | Step-out | 0 | 1 | 1 | u | h | V | r1 | r0 |
| II | Read Sector | 1 | 0 | 0 | m | h | E | 0 | 0 |
| II | Write Sector | 1 | 0 | 1 | m | h | E | P | a0 |
| III | Read Address | 1 | 1 | 0 | 0 | h | E | 0 | 0 |
| III | Read Track | 1 | 1 | 1 | 0 | h | E | 0 | 0 |
| III | Write Track | 1 | 1 | 1 | 1 | h | E | P | 0 |
| IV | Force Interrupt | 1 | 1 | 0 | 1 | I3 | I2 | I1 | I0 |

Diese Kommandos haben jeweils mehrere Flag-Bits, die im einzelnen folgende Bedeutung haben:

**h = Motor On Flag**

h = 0 Motor On-Test einschalten
h = 1 Motor On-Test ausschalten

Wird der Laufwerksmotor eingeschaltet, so sollte eine Wartezeit eingelegt werden, bis der Motor seine Nenndrehzahl erreicht hat. Dies wird vom WD1772 so gehandhabt, daß er nach dem Einschalten des Motors 6 Index-Impulse abwartet. Bei einer Nenndrehzahl des Motors von 300 UPM beträgt diese Wartezeit mindestens eine Sekunde. Dieser Vorgang ( Spin-Up- Sequenz genannt) soll sicherstellen, daß die Motoren ihre Solldrehzahl erreicht haben, wenn die Schreib/Lese-Operationen stattfinden.

Nach Beendigung eines Kommandos werden die Laufwerksmotoren erst nach 10 weiteren Umdrehungen der Diskette (ca. 2 s) ausgeschaltet.

Folgt in der Nachlaufphase ein weiteres Kommando, wäre es natürlich reine Zeitverschwendung erneut eine Spin-Up-Sequenz einzulegen. Aus diesem Grunde wurde in den Controller ein Motor-On-Test implementiert.Ist dieser eingeschaltet (h=0), so wird erst einmal der Motor-On-Ausgang getestet. Nur falls Motor-On auf LOW liegt, legt der FDC die zuvor beschriebene Spin-Up-Sequenz ein. Liegt Motor-On jedoch auf High, so nimmt der Controller an, daß die Motoren mit Solldrehzahl laufen und fährt mit der Abarbeitung seines Kommandos fort.

Wenn der FDC ein Kommando (mit gesetztem h-Bit) empfängt, schaltet er zunächst die Laufwerksmotoren ein, indem er den MOTOR-ON-Ausgang auf HIGH legt. Dies ist unabhängig davon, ob der MOTOR-ON-Ausgang vielleicht schon auf High liegt. Danach beginnt er sofort mit der Ausführung des Kommandos. Er wartet also nicht bis 6 Index-Impulse eingetroffen sind.

**V = Verify Flag**

V = 0 Verify ausschalten
V = 1 Verify einschalten

Dieses Flag-Bit existiert nur in der Gruppe der Typ-I-Kommandos. Ist es gesetzt, so nimmt der Controller, nach einem Step-Kommando bzw. nach dem letzten Step in einem Restore- oder Seek-Kommando, eine Spurverifizierung vor. Dies geschieht in der Form, daß nach dem Step ein korrektes ID-Feld gesucht wird, dessen Spur-Nummer mit dem Inhalt des Spur-Registers übereinstimmt.

Ob man nun ein Kommando mit oder ohne Verify ausführt, sollte von dem nachfolgenden Kommando abhängig gemacht werden, da ein Verify nicht unbedingt nötig und auch nicht immer sinnvoll ist.

Folgt ein READ- oder WRITE-SECTOR Befehl und der Schreib/Lese-Kopf befindet sich nicht über der gewünschten Spur, so wird keinesfalls ein falscher Sektor gelesen oder geschrieben, da bei diesen Kommandos generell ein Verify ausgeführt wird, welches außerdem noch etwas ausführlicher ist. Hier erübrigt sich also ein STEP-Befehl mit Verify.

Völlig unsinnig ist ein Verify beispielweise, wenn eine neue Diskette formatiert wird. Das Verify nach jedem Step-Befehl würde sowieso negativ ausfallen. Da der Controller allerdings für die Zeit von 5 Umdrehungen nach einem korrekten ID-Feld sucht, werden beim Formatieren von 80 Spuren ca.1½ Minuten kostbarer Zeit sinnlos vergeudet.

Wird auf einer schon mit Daten beschriebenen Diskette nachträglich eine einzelne Spur neu formatiert, dann sollte sicherheitshalber ein Verify veranlaßt werden. Denn der WRITE-TRACK-Befehl, der ja zum Formatieren einer Spur verwendet wird, nimmt vor seiner Ausführung keinerlei Tests an der Spur vor. Es wird also die Spur formatiert, über der sich der Schreib/Lese-Kopf gerade befindet.

**r1, r0 = Stepping Rate**

1 0 2ms
0 1 3ms
1 0 5ms
1 1 6ms

Mit diesen beiden Bits läßt sich die Stepping-Rate programmieren. Das ist die Verzögerungszeit, die bei einem SEEK- bzw. RESTORE-Kommando zwischen den einzelnen Step-Impulsen eingelegt wird. Man hat dadurch die Möglichkeit, den FDC in gewissen Grenzen an die mechanischen Gegebenheiten des Laufwerks anzupassen.

Nehmen wir als Beispiel einmal ein Laufwerk, dessen Kopfmechanik für einen Schritt 6 ms benötigt. Wenn nun die Stepping-Rate auf 3 ms programmiert ist und der Kopf durch ein Seek-Kommando von Spur 0 auf Spur 40 bewegt werden soll (was 39 Step-Impulsen entspricht), so würde der Kopf nur die Spur 20 erreichen, da aufgrund der mechanischen Trägheit, jeder zweite Impuls "verschluckt" wird.

Mit einzelnen Step-Impulsen (Step, Step-in, Step-out) wird der Kopf wahrscheinlich korrekt gesteuert, da die Dauer der Step-Impulse durch die Stepping-Rate nicht beeinflußt wird. Durch das interne Timing des FDC bestimmt, beträgt diese einheitlich 4 Mikrosekunden.

**u = Update Flag**

u = 0 kein Update des Spur-Registers
u = 1 Update des Spur-Registers

Ist bei einem STEP-, STEP-IN- oder STEP-OUT-Kommando das u-Bit gesetzt, so wird nach der Operation das Spur-Register um 1 erhöht bzw. um 1 vermindert. Das bedeutet allerdings

nicht, daß der Inhalt des Spur-Registers mit der tatsächlichen Spur übereinstimmt. Hierzu müssen zwei Voraussetzungen erfüllt sein:

1. Die tatsächliche Spur-Nummer muß vor dem Step mit dem Inhalt des Spur-Registers übereingestimmt haben.

2. Es darf nicht versucht werden eine Spur-Nummer > 82 anzusteuern. Wenn sich der Schreib/Lese-Kopf z.B. auf der Spur 82 (die letzte vom Laufwerk erreichbare Spur) befindet und auch der Inhalt des Spur-Registers 82 beträgt, so würde nach 5 STEP-IN Kommandos das Spur-Register die Spur-Nummer 87 enthalten, während sich der Kopf immer noch auf Spur 82 befindet.

**m = Multiple Sector**

m = 0 einen Sektor lesen bzw. schreiben
m = 1 mehrere Sektoren lesen bzw. schreiben

Durch dieses Bit hat man die Möglichkeit, mit nur einem READ- bzw. WRITE-SECTOR-Befehl, maximal alle Sektoren der Spur zu lesen oder zu schreiben. Voraussetzung dazu ist aber, daß die Sektor-Nummern eine lückenlose Reihenfolge bilden. Die Nummer des ersten zu lesenden bzw. zu schreibenden Sektors wird zuvor in das Sektor-Register geschrieben. Nachdem der FDC diesen Sektor gelesen bzw. geschrieben hat, erhöht er das Sektor-Register und versucht den nächsten Sektor zu lesen bzw. zu schreiben. Dies setzt sich solange fort, bis kein weiterer Sektor mehr gefunden wird bzw. das Kommando durch einen FORCE-INTERRUPT-Befehl beendet wird.

**a0 = Data-Address-Mark**

a0 = 0 normales Data Mark schreiben ($FB)

a0 = 1 gelöschtes Data Mark schreiben ($F8)

Das Data-Address-Mark kennzeichnet den Beginn des Datenfeldes. Da durch das a0-Bit (je nachdem ob es gesetzt oder gelöscht ist) unterschiedliche Data-Address-Marks geschrieben werden können, ist man in der Lage, einen Sektor auf einfache Weise zu kennzeichnen. Nach einem READ SECTOR Kommando wird die Art des Data-Address-Mark's im RECORD-TYPE-Statusbit angezeigt.

**E = 30ms Settling Delay**

E = 0 keine Kopfberuhigungszeit
E = 1 30ms Kopfberuhigungszeit

Es gibt Laufwerkstypen bei denen der Schreib/Lese-Kopf nicht ständig das Speichermedium berührt. Bei diesen läßt sich der Kopf durch einen Hubmagneten anheben bzw. absenken. Diese, Head Load genannte, Einrichtung soll den Verschleiß der Diskette verringern, indem man den Kopf nur die Diskette absenkt wenn tatsächlich Lese- oder Schreiboperationen erfolgen. Durch dieses Absenken treten jedoch Schwingungen in der Kopf-Mechanik auf, die einen optimalen Kontakt zwischen Speichermedium und Kopf für eine gewisse Zeit verhindern. Durch Setzen des E-Bits läßt sich ein Delay einschalten, welches diese Zeit überbrückt.

**P = Write Precompensation**

P = 0 Schreib-Vorkompensation einschalten
P = 1 Schreib-Vorkompensation ausschalten

**I0-I3 = Interrupt-Bedingungen**

I0 = 1 Keine Bedeutung
I1 = 1 Keine Bedeutung
I2 = 1 Interrupt bei jedem Index-Impuls

I3 = 1 Sofortiger Interrupt
I0-I3 = 0 Laufendes Kommando ohne Interrupt beenden

**Die Typ-I-Kommandos**

Die Typ-2- und Typ-3-Kommandos, die für das Lesen und Schreiben von Daten zuständig sind, beziehen sich immer auf die Spur, über der sich der Schreib/Lese-Kopf momentan befindet. Für dessen Positionierung ist die, aus den Kommandos Restore, Seek, Step, Step-in und Step-out bestehende Gruppe, zuständig.

**RESTORE (Spur 0 suchen)**

| Kommando-Wort: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | h | V | r1 | r0 |

Wird dem FDC dieses Kommando übergeben, so testet er zunächst den TR00-Eingang. Sollte sich dieser auf LOW befinden, da der Schreib/Lese-Kopf bereits über der Spur Null steht, so wird lediglich das Spur-Register auf Null gesetzt.

Steht der Schreib/Lese-Kopf nicht über der Spur Null, so werden solange STEP-OUT Impulse erzeugt bis der TR00-Eingang auf LOW ist. Sollte dies nach 255 Step-Impulsen noch nicht der Fall sein, so wird das Kommando abgebrochen. Die Beendigung des Kommandos wird durch Setzen des INTRQ-Ausgangs und Löschen des BUSY-Statusbits angezeigt.

**SEEK (Spur suchen)**

| Kommando-Wort: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 1 | h | V | r1 | r0 |

Mit diesem Befehl kann man den Schreib/Lese-Kopf direkt über eine bestimmte Spur steuern. Dazu wird die gewünschte Spur-Nummer ins Daten-Register geladen. Voraussetzung für ein ordnungsgemäßes Funktionieren dieses Befehls ist, daß sich die aktuelle Spur-Nummer im Spur-Register befindet. Werden mehrere Laufwerke betrieben, muß also eventuell auch das Spur-Register geladen werden, damit in diesem auch tatsächlich die aktuelle Spurnummer steht. Erhält der FDC ein SEEK-Kommando, so vergleicht er das Spur-Register mit dem Daten-Register, wodurch er feststellt, ob Step-in oder Step-out Impulse erforderlich sind. Danach werden Step-Impulse für die entsprechende Richtung ausgegeben. Ein UPDATE des Spur-Registers erfolgt nach jedem Step-Impuls. Sobald die Inhalte von Spur- und Daten-Register gleich sind, ist die Ziel-Spur erreicht und das Kommando beendet. Dies wird durch Setzen des INTRQ-Ausgangs und löschen des BUSY-Statusbits angezeigt.

## STEP

| Kommando-Wort: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 1 | u | h | V | r1 | r0 |

Dieser Befehl veranlaßt den FDC einen Step-Impuls auszugeben. Die Richtung ist dabei die gleiche, wie bei einem vorangegangenen Step-Impuls, da der Zustand des DIRECTION-Ausgangs nicht verändert wird. Der INTRQ-Ausgang wird danach auf HIGH gelegt und das BUSY-Statusbit gelöscht.

## STEP-IN

| Kommando-Wort: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 0 | u | h | V | r1 | r0 |

Bei einem Step-In-Kommando wird der DIRECTION-Ausgang, unabhängig vom vorherigen Zustand, auf HIGH gelegt und ein Step-Impuls ausgesendet. Durch diesen wird der Schreib/Lese-

Kopf einen Schritt in Richtung Diskettenmitte bewegt. Der INTRQ-Ausgang wird auf HIGH gelegt und das BUSY-Statusbit gelöscht.

## STEP-OUT

| Kommando-Wort: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 1 | u | h | V | r1 | r0 |

Bei diesem Befehl wird der DIRECTION-Ausgang, unabhängig vom vorherigen Zustand, auf LOW gelegt, bevor der Step-Impuls ausgegeben wird. Der Schreib/Lese-Kopf wird durch den Impuls einen Schritt in Richtung Diskettenrand bewegt. Der INTRQ-Ausgang wird danach auf HIGH gelegt und das BUSY-Statusbit gelöscht.

### Die Verify Sequenz bei den Typ-I-Kommandos

Wurde im Kommandowort das V-Bit gesetzt, so wird, wenn die gewünschte Spur erreicht ist, folgende Sequenz, nach Einlegen eines 30ms Delay's, durchlaufen:

Die Spur-Nummer aus dem ersten gelesenen ID-Feldes wird mit dem Inhalt des Spur-Registers verglichen. Bei Übereinstimmung werden die CRC-Bytes des ID-Feldes getestet. Sollten diese mit den durch die CRC-Logik berechneten übereinstimmen, so ist die Verify-Sequenz fehlerfrei abgeschlossen. Der INTRQ-Ausgang wird auf HIGH gelegt und das BUSY-Statusbit gelöscht.

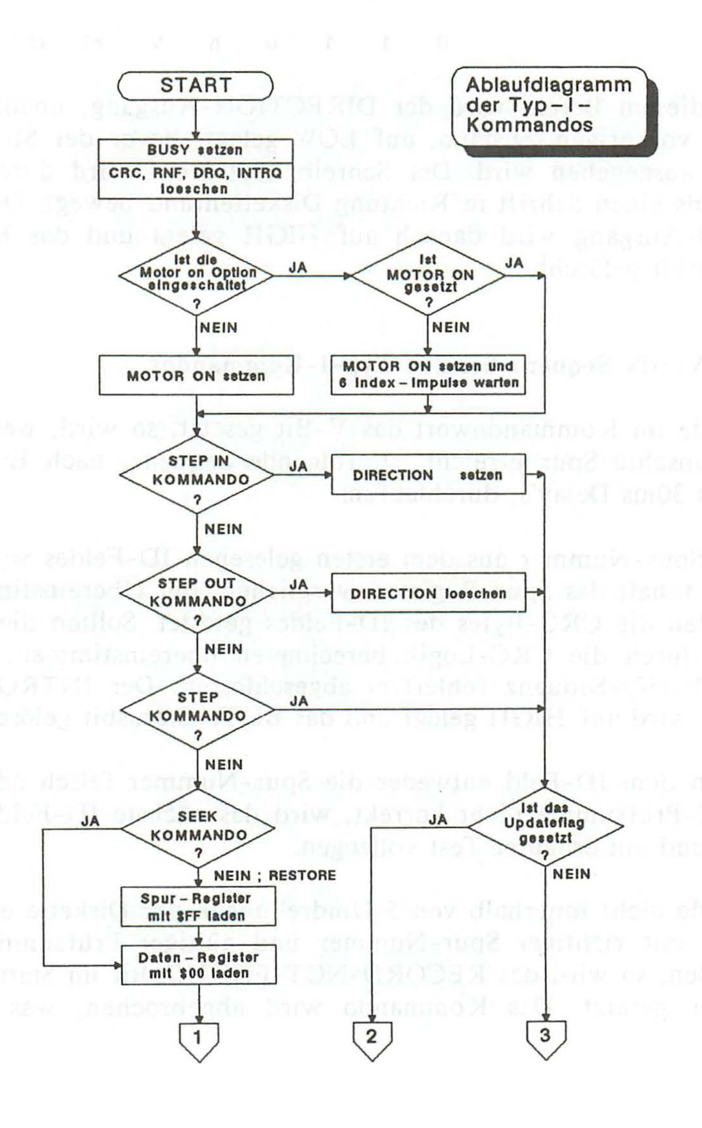
Ist in dem ID-Feld entweder die Spur-Nummer falsch oder die CRC-Prüfsumme nicht korrekt, wird das nächste ID-Feld gelesen und ein erneuter Test vollzogen.

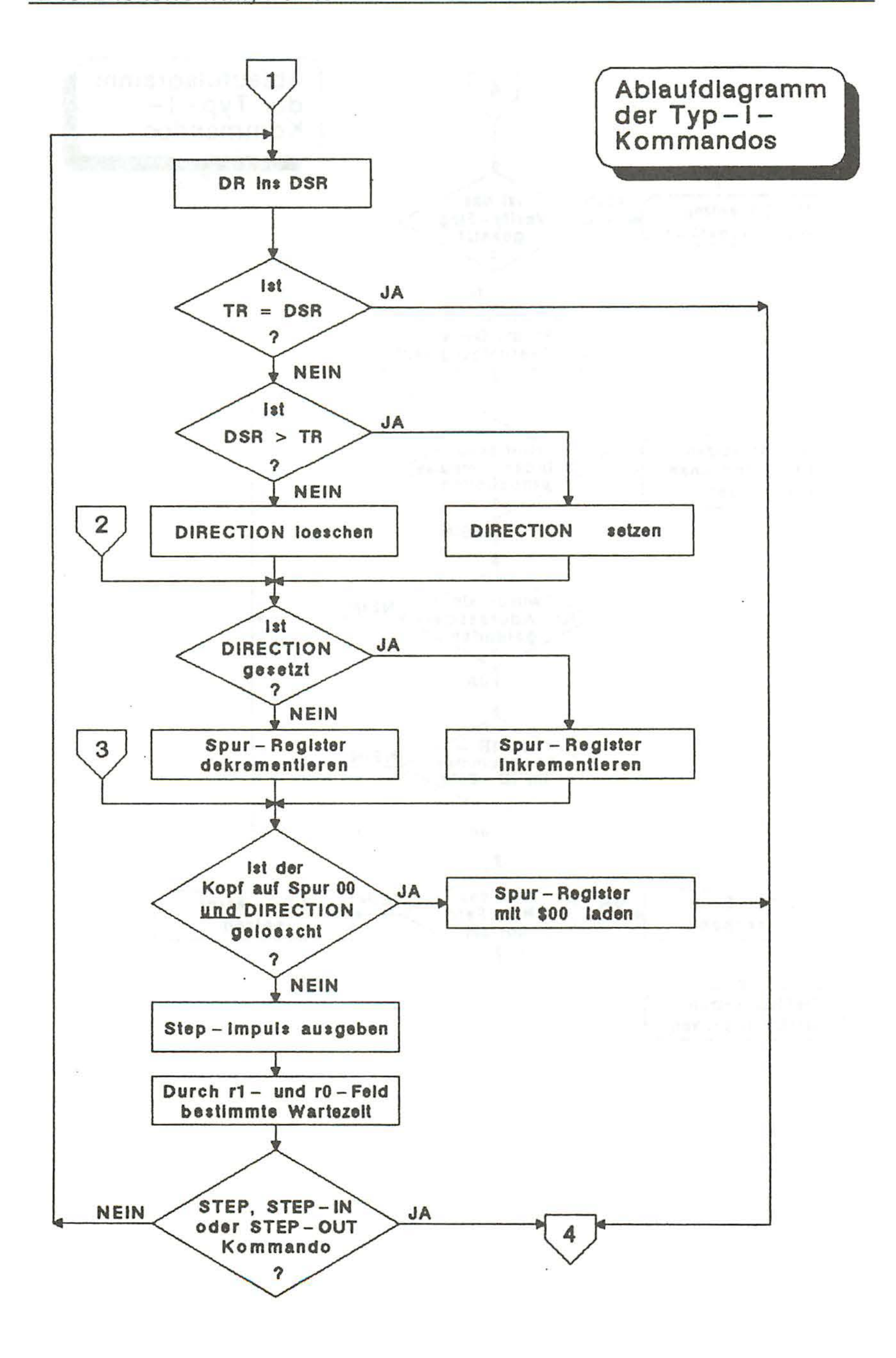
Wurde nicht innerhalb von 5 Umdrehungen der Diskette ein ID-Feld mit richtiger Spur-Nummer und gültiger Prüfsumme gefunden, so wird das RECORD-NOT-FOUND-Bit im Status-Register gesetzt. Das Kommando wird abgebrochen, was durch

setzen des INTRQ-Ausgang und löschen des BUSY-Statusbit angezeigt wird.

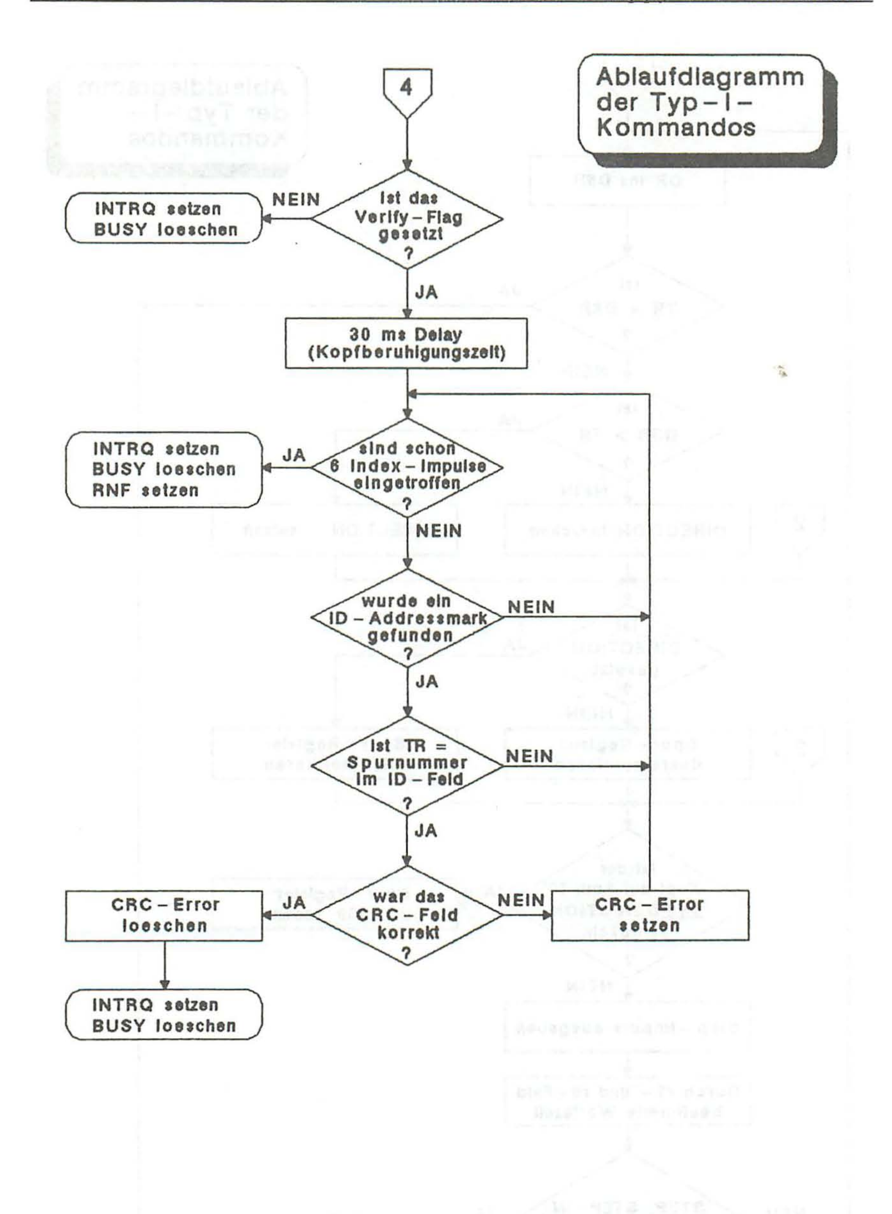
### Ablaufdiagramm der Typ-I-Kommandos

Um den Ablauf der Typ-I-Kommandos deutlicher zu machen, liefern wir ihnen hierzu das entsprechende Ablauf-Diagramm.





Ablaufdiagramm der Typ-I-Kommandos
1
DR ins DSR
Ist TR = DSR ?
JA
NEIN
Ist DSR > TR ?
JA
NEIN
2
DIRECTION loeschen
DIRECTION setzen
Ist DIRECTION gesetzt ?
JA
NEIN
3
Spur-Register dekrementieren
Spur-Register inkrementieren
Ist der Kopf auf Spur 00 und DIRECTION geloescht ?
JA
Spur-Register mit $00 laden
NEIN
Step-Impuls ausgeben
Durch r1- und r0-Feld bestimmte Wartezeit
STEP, STEP-IN oder STEP-OUT Kommando ?
NEIN
JA
4


Ablaufdiagramm der Typ-I-Kommandos
4
Ist das Verify-Flag gesetzt ?
NEIN
INTRQ setzen BUSY loeschen
JA
30 ms Delay (Kopfberuhigungszeit)
sind schon 6 Index-Impulse eingetroffen ?
JA
INTRQ setzen BUSY loeschen RNF setzen
NEIN
wurde ein ID-Addressmark gefunden ?
NEIN
JA
Ist TR = Spurnummer im ID-Feld ?
NEIN
JA
war das CRC-Feld korrekt ?
JA
NEIN
CRC-Error loeschen
CRC-Error setzen
INTRQ setzen BUSY loeschen

**Die Typ-II-Kommandos**

Diese Gruppe ist für das Lesen und Schreiben von Sektoren, welche die logische Dateneinheit darstellen, zuständig. Der Datenaustausch wird deshalb ausschließlich mit diesen Kommandos abgewickelt. Wie alle Kommandos, die für das Lesen und Schreiben von Informationen zuständig sind, nehmen auch diese immer auf die Spur Bezug, über der sich der Schreib/Lese-Kopf gerade befindet.

Daher ist es Aufgabe des Programmierers, mit Hilfe der Typ-I-Kommandos dafür zu Sorgen, daß der Kopf vor Anwendung eines Schreib/Lese-Befehls über der entsprechenden Spur positioniert wird.

## READ-SECTOR

| Kommando-Wort: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | 1 | 0 | 0 | m | h | E | 0 | 0 |

Dem FDC wird die Nummer des zu lesenden Sektors im Sektor-Register übergeben. Wird dann ein READ-SECTOR-Kommando gestartet, so liest der Controller ein ID-Feld und testet, ob dessen Spur- und Sektor-Nummer mit den Inhalten von Spur- und Sektor-Register übereinstimmen. Ist das nicht der Fall, so wird das nächste ID-Feld gelesen.

Waren sie jedoch gleich so wird die Angabe der Sektor-Länge zwischengespeichert und die CRC-Prüfsumme des ID-Feldes mit der aus den gelesenen Daten berechneten verglichen. Sollten sich hier keine Abweichungen ergeben, dann wurde das, zu dem gewünschten Sektor 'passende', ID-Feld gefunden; andernfalls wird ein weiteres ID-Feld gelesen.

Bevor nun das Lesen des Daten-Feldes beginnen kann, muß sich unter den nächsten 43 Byte ein DATA-ADDRESS-MARK (DAM) befinden. Wenn nicht, so wird (wie sollte es anders sein) wiederum ein ID-Feld gelesen.

Wurde nach 5 Umdrehungen kein 'passendes' ID-Feld gefunden, dem nicht innerhalb von 43 Byte ein DAM folgt, so wird das RNF-Statusbit gesetzt und das Kommando abgebrochen.

Wie Sie sehen, müssen schon einige Bedingungen erfüllt sein, bevor der FDC ein Daten-Feld, in dem sich ja ein Sektor befindet, liest. Ist bis hierhin aber alles ordnungsgemäß verlaufen, dann geht es wie folgt weiter:

Die Art des DATA-ADDRESS-MARK's ('normal' = 0; 'gelöscht' = 1) wird in das Status-Bit 5 übertragen. Der Address-Mark-Detector wird ausgeschaltet und die entsprechende Anzahl (aus der Angabe im Sektor-Längen-Feld berechnet) von Daten-Bytes gelesen. Für jedes gelesene Byte wird ein DRQ ausgelöst. Sollte auf einen dieser DRQ's nicht reagiert werden, so setzt der FDC das LOST-DATA-Statusbit.

Nachdem alle Daten-Bytes gelesen sind, wird auch hier eine CRC-Prüfsumme getestet. Diese befindet sich in den beiden, dem Sektor folgenden, Byte und wird mit der, aus den gelesenen Sektor-Daten, berechneten verglichen. Bei Abweichung wird das CRC-ERROR-Statusbit gesetzt.

Das Ende der Operation wird durch Setzen des DRQ-Ausgangs und Löschen des BUSY-Statusbits angezeigt.

**READ-SECTOR (mit gesetztem m-Bit)**

Wird im Kommandowort des READ-SECTOR-Befehls das m-Bit gesetzt, so wird der FDC versuchen mehrere Sektoren (maximal alle der Spur) zu lesen. Die Nummer des Ersten zu lesenden Sektors wird dem Controller im Sektor-Register übergeben. Der Ablauf des Kommandos ist zunächst mit dem zuvor beschriebenen identisch. Nachdem jedoch der Sektor gelesen wurde, erhöht der Controller automatisch das Sektor-Register und startet ein weiteres READ-SECTOR-Kommando. Dies setzt sich solange fort, bis kein weiterer Sektor mehr gefunden wird.

Das heißt mit anderen Worten, daß dieses Kommando immer mit einem RECORD-NOT-FOUND-ERROR abgebrochen wird.

Da dem Controller nicht die Anzahl der zu lesenden Sektoren angegeben werden kann, muß es eine andere Möglichkeit geben, mehrere Sektoren zu lesen. Dieses erreicht man durch das FORCE-INTERRUPT-Kommando. Es wird also nicht darauf gewartet bis der FDC das Kommando von alleine abbricht, sondern bestimmt diesen Moment selbst.

Wie, das soll dieses Beispiel verdeutlichen:

Voraussetzung: Der Kopf befindet sich über einer beliebigen, nach ATARI-FORMAT formatierten Spur.

Die Sektoren dieser Spur sind demnach von 1-9 numeriert und wir möchten die Sektoren 3-7, also 5 Stück, lesen.

Die Sektor-Nr. 3 wird ins Sektor-Register übertragen und die Lese-Operation durch Übergabe des Kommandowortes gestartet.

Wird nichts weiter unternommen, so würden jetzt 7 Sektoren (Sektor 3-9) gelesen und das Kommando nach 5 weiteren Umdrehungen mit RNF-ERROR abgebrochen (da Sektor-Nr.10 nicht gefunden werden kann).

Es sollen aber nur die Sektoren 3-7 (und zwar ohne Fehlermeldung) gelesen werden. Deshalb kommen wir zu folgendem Ablauf:

Da die Datenübertragung durch den DMA-Controller abgewikkelt wird, wurde dieser ja vor Start des FDC-Kommandos mit einer Anfangs-DMA-Adresse initialisiert. Während der FDC die Daten an den DMA-Controller übergibt, wird diese Adresse fortlaufend erhöht. Die aktuelle DMA-Adresse kann daher durch Auslesen des DMA-Adressen-Registers in Erfahrung gebracht werden. Dieses Register wird nun ständig abgefragt, bis dessen Inhalt gegenüber der Start-Adresse um $A00 (5 * $200) erhöht ist. Sobald das der Fall ist, wird das READ-SECTOR-

Kommando durch einen FORCE-INTERRUPT-BEFEHL abgebrochen.

**WRITE-SECTOR**

| Kommando-Wort: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | 1 | 0 | 1 | m | h | E | P | a0 |

Wir möchten hierzu nur die Unterschiede gegenüber dem READ-SECTOR-Kommando beschreiben, da der größte Teil des Ablaufs mit diesem Übereinstimmt.

Der FDC testet zu Anfang des Kommandos, ob der WRITE-PROTECT-Eingang auf LOW liegt. Sollte das der Fall sein (schreibgeschützte Diskette), so wird das WPRT-Statusbit gesetzt und das Kommando abgebrochen.

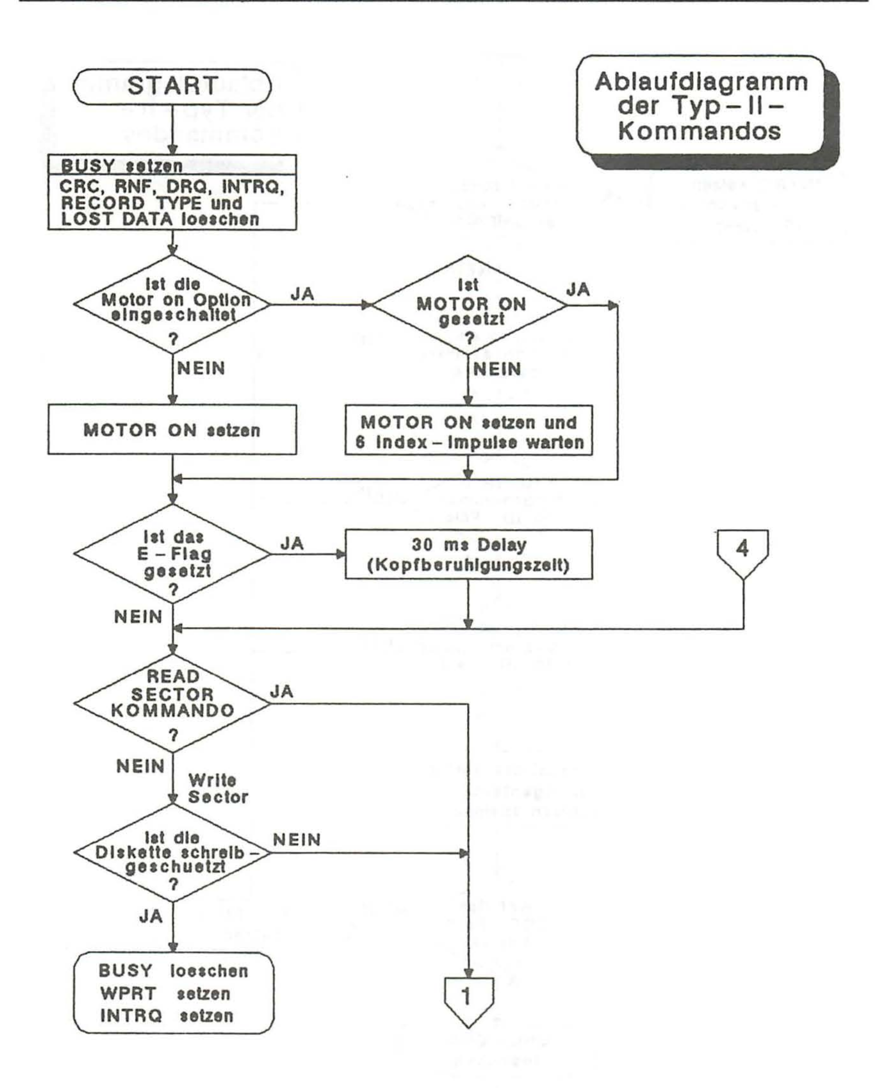
Ist die Diskette nicht schreibgeschützt, dann wird mit der ID-Feld Suche begonnen. Wurde das 'passende' ID-Feld gefunden, wird ein Delay, 23 Byte entsprechend, eingelegt. Danach werden 12 'Null-Bytes' und ein DATA-ADDRESS-MARK (die Art ist vom a0-Bit abhängig) geschrieben. Es folgt die eigentliche Sektor-Information, der sich die CRC-Prüfsumme anschließt. Zu guter Letzt wird noch ein '$FF-Byte' geschrieben. Ob der Sektor richtig geschrieben wurde, kann nur durch ein READ-SECTOR-Kommando festgestellt werden.

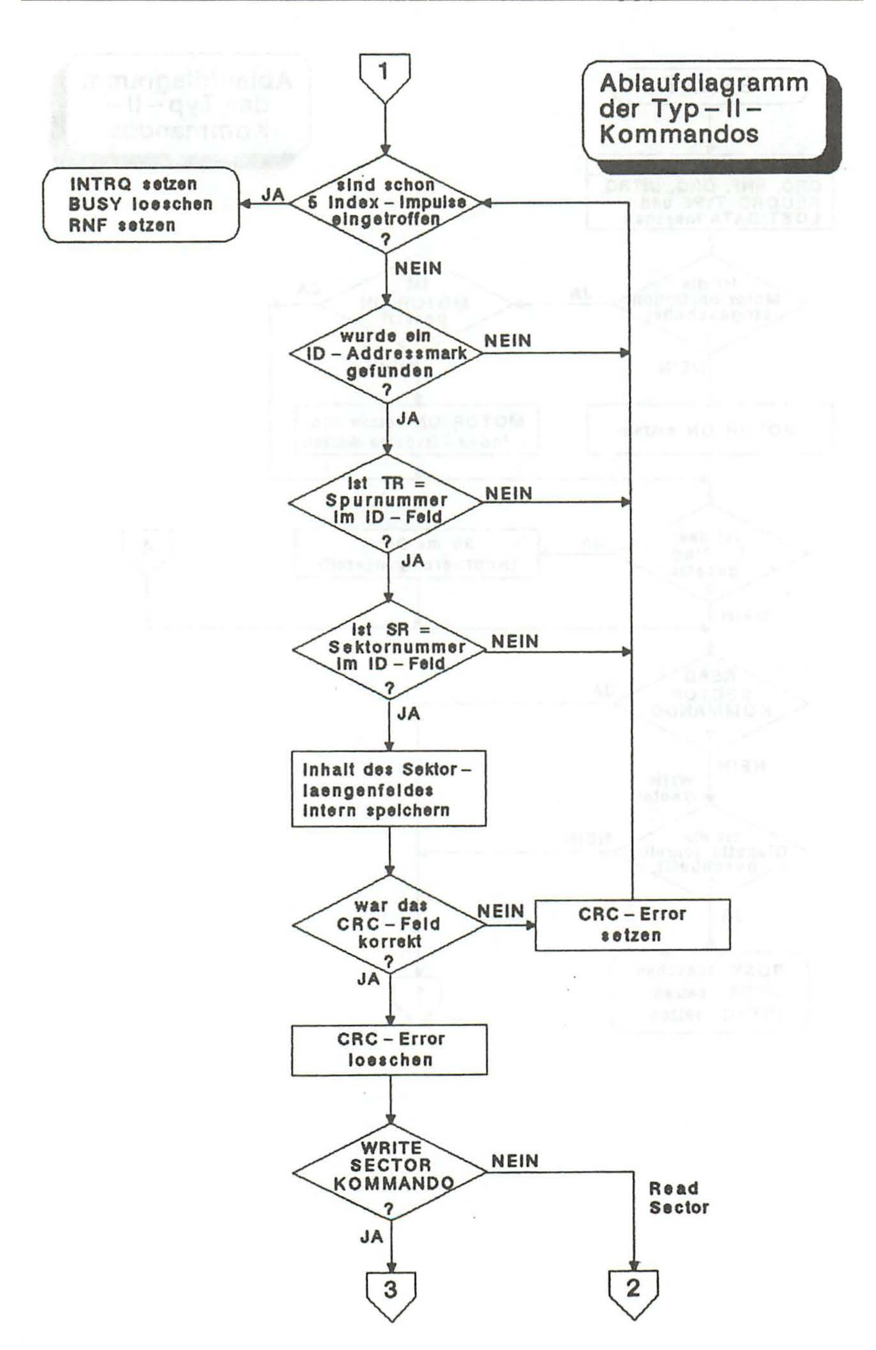
**WRITE-SECTOR (mit gesetztem m-Bit)**

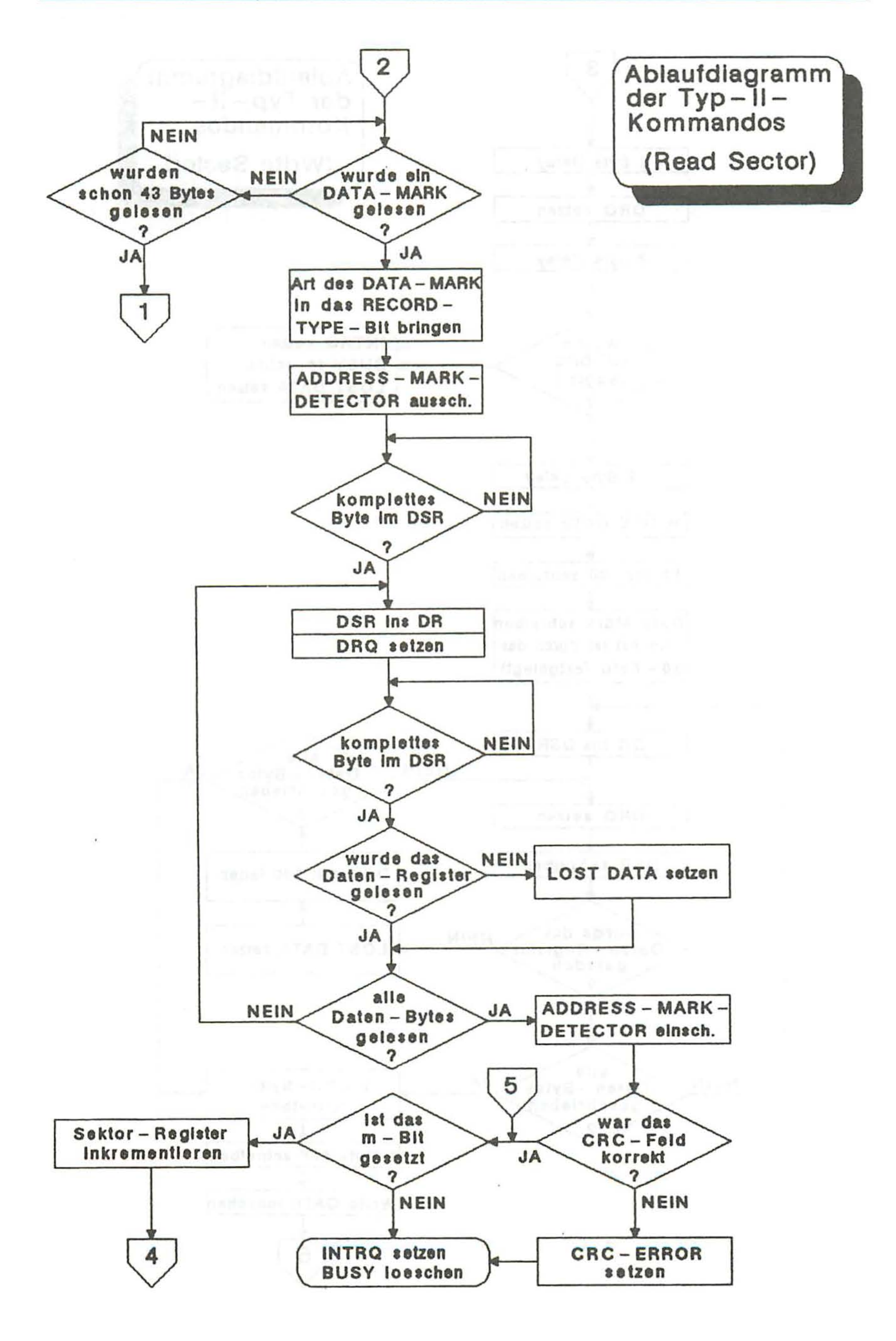
Es gilt hier analog das gleiche wie bei den Ausführungen zum READ-SECTOR-Kommando mit gesetztem m-Bit.

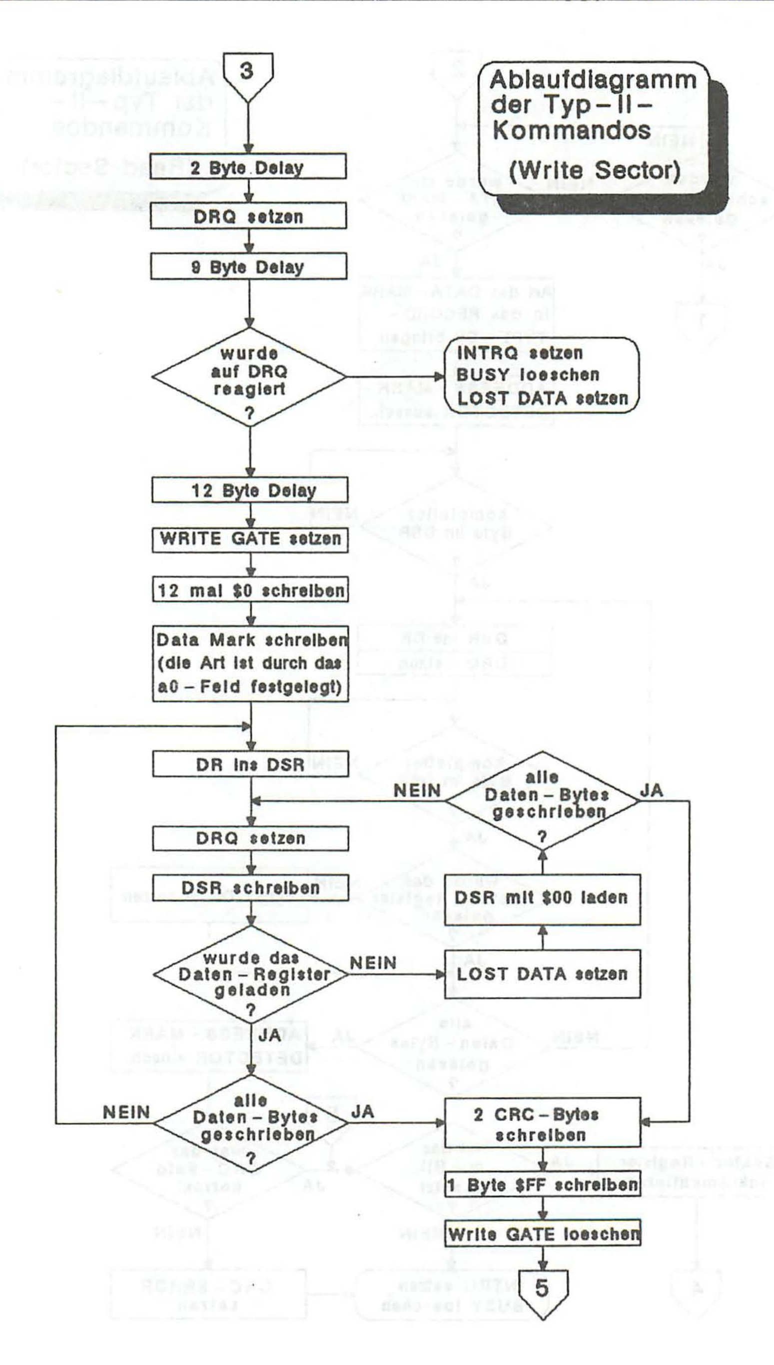
**Ablaufdiagramm der Typ-2-Kommandos**

Auch für die Typ-2-Kommandos können wir ihnen ein Ablauf-Diagramm anbieten.


Ablaufdiagramm
der Typ-II-
Kommandos
START
BUSY setzen
CRC, RNF, DRQ, INTRQ,
RECORD TYPE und
LOST DATA loeschen
Ist die
Motor on Option
eingeschaltet
?
JA
NEIN
Ist
MOTOR ON
gesetzt
?
JA
NEIN
MOTOR ON setzen
MOTOR ON setzen und
6 Index-Impulse warten
Ist das
E-Flag
gesetzt
?
JA
NEIN
30 ms Delay
(Kopfberuhigungszeit)
4
READ
SECTOR
KOMMANDO
?
JA
NEIN
Write
Sector
Ist die
Diskette schreib-
geschuetzt
?
NEIN
JA
BUSY loeschen
WPRT setzen
INTRQ setzen
1


1
Ablaufdiagramm der Typ-II-Kommandos
INTRQ setzen
BUSY loeschen
RNF setzen
JA
sind schon 5 Index-Impulse eingetroffen ?
NEIN
wurde ein ID-Addressmark gefunden ?
NEIN
JA
Ist TR = Spurnummer im ID-Feld ?
NEIN
JA
Ist SR = Sektornummer im ID-Feld ?
NEIN
JA
Inhalt des Sektor-laengenfeldes intern speichern
war das CRC-Feld korrekt ?
NEIN
CRC-Error setzen
JA
CRC-Error loeschen
WRITE SECTOR KOMMANDO ?
NEIN
Read Sector
JA
3
2


Ablaufdiagramm der Typ-II-Kommandos (Read Sector)
2
wurde ein DATA-MARK gelesen ?
NEIN
wurden schon 43 Bytes gelesen ?
NEIN
JA
1
JA
Art des DATA-MARK in das RECORD-TYPE-Bit bringen
ADDRESS-MARK-DETECTOR aussch.
komplettes Byte im DSR ?
NEIN
JA
DSR ins DR
DRQ setzen
komplettes Byte im DSR ?
NEIN
JA
wurde das Daten-Register gelesen ?
NEIN
LOST DATA setzen
JA
alle Daten-Bytes gelesen ?
NEIN
JA
ADDRESS-MARK-DETECTOR einsch.
5
war das CRC-Feld korrekt ?
JA
NEIN
CRC-ERROR setzen
Ist das m-Bit gesetzt ?
JA
NEIN
Sektor-Register inkrementieren
4
INTRQ setzen BUSY loeschen


3
Ablaufdiagramm der Typ-II-Kommandos
(Write Sector)
2 Byte Delay
DRQ setzen
9 Byte Delay
wurde auf DRQ reagiert ?
INTRQ setzen
BUSY loeschen
LOST DATA setzen
12 Byte Delay
WRITE GATE setzen
12 mal $0 schreiben
Data Mark schreiben (die Art ist durch das a0-Feld festgelegt)
DR ins DSR
NEIN
alle Daten-Bytes geschrieben ?
JA
DRQ setzen
DSR schreiben
DSR mit $00 laden
wurde das Daten-Register geladen ?
NEIN
LOST DATA setzen
JA
NEIN
alle Daten-Bytes geschrieben ?
JA
2 CRC-Bytes schreiben
1 Byte $FF schreiben
Write GATE loeschen
5

**Die Typ-III-Kommandos**

Der in dieser Gruppe enthaltene WRITE-TRACK-Befehl dient dem Formatieren einer Spur, während mittels READ-TRACK und READ-ADDRESS das Format einer Spur analysiert werden kann.

**READ ADDRESS**

| Kommando-Wort: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | 1 | 1 | 0 | 0 | h | E | 0 | 0 |

Zu diesem Kommando müssen unbedingt ein paar erklärende Worte, die den DMA-Controller betreffen, vorausgeschickt werden. Es könnte sonst leicht passieren, daß Sie bei der Programmierung des READ-ADDRESS-Befehls an den Rand des Wahnsinns getrieben werden.

Wird das READ-ADDRESS-Kommando gestartet, so stellt man nach dessen Ausführung nämlich erstaunt fest, daß sich das ID-Feld nicht im RAM befindet.

Ein Test der Status-Register (DMA und FDC) wird allerdings keinen Fehler anzeigen. Wenn man nun die Start-DMA-Adresse von der End-DMA-Adresse subtrahiert, dann ergibt die Differenz Null. Es sind also nur 0 Bytes übertragen worden. Das ist natürlich etwas wenig.

Wo sind also die 6 Byte des ID-Feldes geblieben? Nun, ganz einfach, im DMA-Controller! Dieser überträgt nämlich die Bytes nicht einzeln sondern wartet, bis er 16 Byte erhalten hat. Erst dann stellt er eine Busanforderung an den 68000-Prozessor, um diese 16 Byte in das RAM übertragen zu können.

Wie kommt man nun an das gelesene ID-Feld, welches sich ja noch im DMA-Controller befindet? Hier gibt es nur eine Möglichkeit. Es müssen weitere Daten gelesen werden. Dies wird da-

durch erreicht, indem mehrere ID-Felder hintereinander gelesen werden.

Nach z.B. 3 READ-ADDRESS-Befehlen (18 Byte) befinden sich dann 16 Byte im RAM und 2 Byte im DMA-Controller. Damit alle gelesenen Bytes ins RAM übertragen werden, muß deren Anzahl demzufolge ein ganzzahliges Vielfaches von 16 sein.

Es gibt aber noch einen weiteren Fallstrick. Dieser besteht im Löschen des DMA-Statusregisters, welches durch "toggeln", also dem Ein- und Ausschalten, der Schreib/Lese-Leitung erreicht wird. Wer, vor jedem DMA-Transfer, "sicherheitshalber" dieses Register zurücksetzt, erlebt die nächste Überraschung. Hierdurch wird nämlich nicht nur das Status-Register gelöscht, sondern auch alle Bytes, die sich noch im DMA-Controller befinden. In diesem Fall könnten solange ID-Felder gelesen werden, bis die Diskette durchgeschliffen ist. Der DMA-Controller würde nicht ein einziges Byte in den Speicher übertragen. Das DMA-Statusregister darf also nur vor dem ersten READ-ADDRESS-Befehl gelöscht werden.

Doch nun zum READ-ADDRESS-Kommando selbst. Durch dieses Kommando wird jeweils das nächste ID-Feld, das der Schreib/Lese-Kopf erreicht, gelesen. Es kann in Verbindung mit dem READ-TRACK-Kommando dazu benutzt werden, das Format einer Spur zu analysieren. Ferner ist es auch möglich, ein Spur-Verify vorzunehmen, ohne die Spur zu verlassen.

Das READ-ADDRESS-Kommando liest ein ID-Feld, ohne zu testen, ob ein zugehöriges Daten-Feld existiert. Es werden dabei 6 Byte gelesen, die folgende Bedeutung haben:

| Byte-Nr. | Bedeutung |
|---|---|
| 1 | Spur-Nummer |
| 2 | Seiten-Nummer |
| 3 | Sektor-Nummer |

| | |
|---|---|
| 4 | Sektorgröße |
| 5 | CRC-Byte 1 |
| 6 | CRC-Byte 2 |

Die Spur-Nummer (Byte 1) wird außerdem in das Sektor-Register geschrieben. Man kann dadurch also ein Spur-Verify vornehmen, ohne die gelesenen Daten zu verwenden. Dies erscheint im ersten Moment als unnötig, da es ja keine Rolle spielt, ob man für ein Spur-Verify den Inhalt des Sektor-Registers oder das erste übertragene Byte, mit dem Inhalt des Spur-Registers, vergleicht. Außerdem ist es einfacher das übertragene Byte zu verwenden, da in diesem Fall nicht erst das Sektor-Register selektiert werden muß.

Nach einem einzelnen READ-ADDRESS-Befehl wird jedoch noch keine Information ins RAM übertragen, da, wie zuvor schon beschrieben, der Speicher-Transfer erst dann beginnt, wenn der DMA-Controller 16 Byte 'gesammelt' hat. Doch dadurch, daß der FDC das erste Byte des ID-Feldes noch zusätzlich in das Sektor-Register schreibt, hat man trotzdem eine Möglichkeit, durch ein einziges READ-ADDRESS-Kommando, ein Spur-Verify durchzuführen.

Bei dem Lesen des ID-Feldes wird eine Prüfsumme gebildet, die mit der Aufgezeichneten (CRC-Byte 1 und 2) verglichen wird. Sollten diese nicht identisch sein, wird das CRC-ERROR-Statusbit gesetzt.

Ist bis zum Eintreffen von 6 Index-Impulsen (das entspricht mindestens 5 Umdrehungen), kein ID-Feld gefunden worden, so wird das RNF-Statusbit (Record not found) gesetzt.

Hat der Controller das Kommando abgearbeitet, so zeigt er dies durch Setzen des INTRQ-Ausgangs und Löschen des BUSY-Statusbits an.

## READ TRACK

```
Kommando-Wort:   7   6   5   4   3   2   1   0

                 1   1   1   0   h   E   0   0
```

Auch das READ-TRACK-Kommando dient nur der Spur-Diagnose. Es wird eine komplette Spur, inklusive aller GAP-, SYNC- und Daten-Bytes, gelesen.

Das Lesen beginnt mit der steigenden Flanke des nächsten Index-Impulses, den der Controller vom Laufwerk erhält. Es werden solange Daten gelesen, bis ein weiterer Index-Impuls den Controller erreicht. Das Ende der Operation wird, wie üblich, durch Setzen des INTRQ-Ausgangs und Löschen des BUSY-Statusbits angezeigt.

Für jedes gelesene Byte wird ein DRQ erzeugt. Wie bei allen Kommandos die Daten transferieren, wird auch hier das LOST-DATA-Statusbit gesetzt, wenn nicht auf den DRQ reagiert wird.

Bezüglich des LOST-DATA-Bits stellten wir fest, daß es des öfteren, aus uns unbekannten Gründen, gesetzt wird. Das hat zur Folge, daß dieses Bit nach einem READ-TRACK-Kommando keinen Aufschluß darüber gibt, ob tatsächlich Daten verloren gegangen sind. Merkwürdigerweise traten solche Fälle bei READ-TRACK-Versuchen an einer unformatierten Diskette nie auf.

Mit jedem empfangenen Addreß-Mark wird das 'Sammeln' der Daten-Bits synchronisiert. Der ADDRESS-MARK-DETECTOR, der hierfür zuständig ist, wird danach aber nicht (wie z.B bei einem READ-SECTOR-Kommando) ausgeschaltet, sondern bleibt während des gesamten Lesevorgangs aktiviert. Er ist ständig auf der Suche nach einem Addreß-Mark und verursacht dadurch Lesefehler.

Nach Herstellerangaben sollen alle Informationen, mit Ausnahme der GAP-Bytes, korrekt gelesen werden. Unsere Versuche haben allerdings andere Ergebnisse erbracht. In der Praxis ist es so,

daß nur die ID-Felder richtig gelesen werden. Aber schon bei der CRC-Prüfsumme der ID-Felder treten zeitweilig Lesefehler auf.

Es erhebt sich die Frage, wozu das READ-TRACK-Kommando eigentlich verwendet werden kann. Die Daten selbst sind ja durch mögliche Lesefehler nur von zweifelhaftem Wert und korrekt gelesene ID-Felder erhält man durch das READ-ADDRESS-Kommando wesentlich einfacher, denn die Suche nach einem ID-Feld in der gesamten Spur-Information wirft ein weiteres Problem auf. So läßt sich bei einer Bytefolge von FE-01-00-01-02-BC-DB nicht erkennen, ob es sich hier tatsächlich um ein ID-Feld handelt. Diese Bytefolge könnte ja auch in einem Daten-Feld vorkommen. Auch wenn kein ID-Feld existiert könnte Sie zufällig auf der Diskette stehen. Ein 'echtes' ID-Feld ist nur ein solches, welches der Controller als ID-Feld erkennt.

Der READ-TRACK-Befehl ist, alleine angewandt, nicht sehr sinnvoll. Im Zusammenwirken mit dem READ-ADDRESS-Kommando läßt sich eine Spur jedoch ziemlich genau analysieren. Wenn die Daten selbst zum größten Teil keine Aussagekraft besitzen, so ist deren Anzahl jedoch von großer Bedeutung. Man kann dadurch die Abstände zwischen 'markanten' Punkten, was für die Spuranalyse wichtig ist, sehr genau berechnen. Hierdurch wird der Nachteil des READ-ADDRESS-Kommandos, welches nur ID-Felder liest, aber nicht testet ob ein zugehöriges Daten-Feld existiert, ausgeglichen.

Als Beispiel möchten wir im Ansatz den Ablauf einer Spuranalyse schildern:

a. Alle ID-Felder der Spur werden durch READ-ADDRESS-Kommandos gelesen.

b. Die gesamte Spur-Information wird mit einem READ-TRACK-Befehl gelesen.

c. Alle Sektoren der Spur (Spur- und Sektor-Nummer erhält man aus den gelesenen ID-Feldern) werden durch READ-SECTOR-Kommandos gelesen.

Sind in der Spur keine ID-Felder enthalten, so ist unsere Analyse schon beendet, da wir es dann mit einem unlesbaren Format zu tun haben.

Es wird nun in der Spur-Information nach dem ersten ID-Feld gesucht. Hier muß man sich an den eben erwähnten 'markanten' Punkten orientieren.

Der erste dieser Punkte ist eine Bytefolge von $A1,$FE oder $C2,$FE, also SYNC-Byte und ID-ADDRESS-MARK. Nur einem solchen Punkt kann ein ID-Feld folgen. Wurde es gefunden, so wird der zweite 'markante' Punkt interessant. Dieser befindet sich max. 42 Byte hinter dem ID-Feld. Es muß hier ein SYNC-Byte, gefolgt von einem DATA-ADDRESS-MARK gefunden werden. Falls nicht, so existiert für das ID-Feld, kein gültiges Daten-Feld.

Ein weiterer Test betrifft die Plausibilität des ID-Feldes. Zeigt dieses z.B. eine Sektor-Größe von 512 Byte an und das nächste ID-Feld folgt in einem Abstand von 200 Byte, so ist hier etwas im Argen.

Das Lesen der Sektoren erfolgt in diesem Zusammenhang aus zwei Gründen. Zum einen kann die Sektor-Information (wegen der schon beschriebenen Datenverfälschungen) nicht den, durch READ-TRACK erhaltenen Daten entnommen werden. Zum anderen ist nur durch READ-SECTOR eine Kontrolle der CRC-Prüfsumme des Datenfeldes möglich.

Auf diese oder ähnliche Weise wird der gesamte Spur-Inhalt analysiert. Die Auswertung der daraus erhaltenen Informationen gibt die Möglichkeit, die soeben analysierte Spur zu reproduzieren bzw. die Erkenntnis, daß sich die Spur mit den Möglichkeiten des WD 1772 nicht reproduzieren läßt.

## WRITE TRACK (FORMAT TRACK)

| Kommando-Wort: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | 1 | 1 | 1 | 1 | h | E | P | 0 |

Bevor eine Diskette zur Datenspeicherung benutzt werden kann, muß diese als erstes formatiert werden. Doch was geschieht dabei eigentlich? Die Sache ist im Grunde genommen sehr leicht zu erklären.

Die logische Dateneinheit, mit der ein Datentransfer zwischen Laufwerk und Controller stattfindet, ist der Sektor. Da sich auf einer neuen Diskette aber keine Informationen über den Startpunkt eines Sektors befinden, muß diese erst mit solchen Startpunkten versehen werden.

Zu jedem Sektor gehört ein Feld, welches Informationen über diesen beinhaltet. Ferner sind die Daten durch Prüfsummen gesichert. Auch das fehlt auf einer neuen Diskette und muß zuvor auf diese geschrieben werden. Synchronisations-Bytes sind dort natürlich auch nicht zu finden. Diese sind jedoch eminent wichtig um den Anfang eines Bytes, welcher sich bei späteren Leseoperationen irgendwo im seriellen Bit-Strom 'versteckt', zu erkennen.

Ziel des Formatierens ist es nun, alle hier aufgeführten Informationen bzw. Markierungen, auf die Diskette zu schreiben. Dies muß jedoch nach bestimmten Regeln geschehen, da später sonst kein Sektor-Transfer möglich ist.

Hält man sich an diese Regeln, so lassen sich zahlreiche - auch vom Standard abweichende - einwandfrei funktionierende, Formate erzeugen.

Dem FDC wird grundsätzlich ein Datenbyte, also ein Wert zwischen $00 und $FF, übergeben. Um aber die erforderlichen Markierungen, die sich von 'normalen' Datenbytes unterscheiden, auf die Diskette zu schreiben, muß es eine Möglichkeit ge-

ben, den Controller so zu steuern, daß nicht ein Datenbyte, sondern eine Markierung auf die Diskette geschrieben wird.

Wir möchten deshalb zunächst die Steuerbytes behandeln, welche durch Werte zwischen $F5 und $FF repräsentiert werden. Im Gegensatz zu den READ-SECTOR- bzw. WRITE-SECTOR-Kommandos, bei welchen diese als 'normale' Daten-Bytes geschrieben werden, veranlassen sie den Controller bei einem WRITE-TRACK-Kommando, die Diskette mit besonderen Markierungen zu versehen. Gemeinsam für diese gilt, daß sie ohne Takt-Impulse geschrieben werden und daher, bei späteren Lese-Operationen von Datenbytes, die eventuell den gleichen Wert besitzen, unterschieden werden können (siehe 'Address-Mark-Detector'). Was diese Steuer-Bytes, bei einem WRITE-TRACK-Befehl im einzelnen bewirken bzw. welche Bedeutung sie dadurch bei späteren Lese-Operationen erlangen, zeigt die folgende Übersicht:

| an den FDC übergebenes Byte | vom FDC geschriebene(s) Byte(s) | Bedeutung |
|---|---|---|
| $F5 | $A1 | Sync-Byte, CRC-Reg. löschen |
| $F6 | $C2 | Sync-Byte |
| $F7 | $XX,$XX | 2 CRC-Bytes |
| $F8 | $F8 | 'gelöschtes' Data-Address-Mark |
| $F9 | $F9 | Data-Mark |
| $FA | $FA | Data-Mark |
| $FB | $FB | 'normales' Data-Address-Mark |
| $FC | $FC | Data-Mark |
| $FD | $FD | Data-Mark |
| $FE | $FE | Index-Address-Mark |
| $FF | $FF | |

Es soll nun erklärt werden, wie mit Hilfe dieser Steuer-Bytes ein beliebiges Format erzeugt werden kann. Da hierzu einige Beispielwerte erforderlich sind, werden jeweils solche benutzt, die zur Erstellung des ATARI-Formates nötig sind. An den ent-

sprechenden Stellen wird darauf hingewiesen, in welchem Maße Abweichungen zulässig sind.

Beginnen wir mit einem Puffer, der alle Informationen einer kompletten Spur, die später durch das WRITE-TRACK-Kommando auf die Diskette geschrieben wird, aufnehmen kann. Dazu sollte dessen Größe mindestens 6250 Byte betragen. Es gilt nun, diesen Puffer derart mit Daten zu füllen, das er einem Format entspricht, welches 9 Sektoren von je 512 Byte Länge aufnehmen kann.

Teilen wir unseren Puffer in 2 verschiedene Komponenten auf, so erhalten wir folgendes Schema, welches für alle Formate Gültigkeit besitzt. Unterschiede in der Anzahl der Records sind natürlich möglich.

```
GAP1    RECORD 1    RECORD 2              RECORD 9    GAP5
```

Eine Spur beginnt und endet mit einem, GAP bezeichneten, Block. Wie wir später noch sehen werden, sind diese GAP's auch in den RECORDs vorhanden. Ein GAP ist ein Zwischenraum, der die einzelnen Komponenten in der Spur trennt. Es enthält keine Nutzinformation, sondern nur Füllbytes bzw. wenn ein GAP vor einem ID- oder Daten-Feld steht auch SYNC-Bytes. Dem FDC wird dadurch eine, durch die Länge des GAPs bestimmte, Zeit eingeräumt, in der er seine Funktionsteile, auf die Erfordernisse der nachfolgenden Komponente, einstellen kann.

| Bezeichnung | Wert | Gap-Länge ATARI-Format | Gap-Länge Fremd-Format |
|---|---|---|---|
| GAP1 (Spur-Vorspann) | $4E | 60 Byte | min. 32 Byte |
| GAP5 (Spur-Nachspann) | $4E | ca.664 Byte | min. 16 Byte |

Die Länge von GAP5 ist im Moment irrelevant. GAP5 ist, einfach formuliert, das, was übrig bleibt. Bei unseren Berechnun-

gen der Puffer-Aufteilung, müssen wir lediglich darauf achten, daß wenigstens 16 Byte für GAP5 vorhanden sind.

Subtrahieren wir von unserem Puffer die Anzahl der Bytes, die wir für GAP1 reservieren, so stehen uns noch 6190 Byte zur Verfügung um sie auf die Anzahl der Records aufzuteilen. Diese Aufteilung läßt sich jetzt noch nicht vornehmen, weil die Länge eines Records noch nicht bekannt ist.

Wie Sie wahrscheinlich schon richtig vermutet haben, ist in jedem Record einer unserer 9 Sektoren enthalten. Deshalb steht eines fest; die Recordlänge ist in jedem Fall größer als die Sektorlänge. Wenn wir einen Record weiter aufschlüsseln, ergibt sich folgendes Bild:

| GAP2 | INDEX-FELD | GAP3 | DATEN-FELD | GAP4 |
|---|---|---|---|---|

Hier sind zunächst die bereits angekündigten GAPs, die in der gezeigten Reihenfolge mit Pre-Record-, Inter-Record- und Post-Record-Gap bezeichnet werden.

| Bezeichnung | | Wert | Gap-Länge ATARI-Format | Gap-Länge Fremd-Format |
|---|---|---|---|---|
| GAP2 | | $00 | 12 Byte | min. 8 Byte |
| | SYNC | $F5 | 3 Byte | 3 Byte |
| GAP3 | | $4E | 22 Byte | 22 Byte |
| | | $00 | 12 Byte | 12 Byte |
| | SYNC | $F5 | 3 Byte | 3 Byte |
| GAP4 | | $4E | 40 Byte | min. 24 Byte |
| Summe der GAP-Bytes pro RECORD: | | | 92 Byte min. | 72 Byte |

Die Synchronisations-Bytes ($F5) in Gap2 und Gap3 sorgen dafür, daß das Lesen, der seriell eintreffenden Daten-Bits, mit dem Byte-Anfang geschieht. Ferner haben sie die Aufgabe, den FDC auf ein folgendes ADDRESS-MARK aufmerksam zu machen und dessen CRC-Logik zu initialisieren. Um ein SYNC zu schreiben wird dem FDC der Wert $F5 übergeben, der daraufhin ein '$A1-Byte', ohne Takt-Impulse schreibt.

**Das Daten-Feld**

Die GAPs, die sich in einem RECORD befinden sind nun aufgeschlüsselt. Sehen wir uns nun das Daten-Feld, in dem sich ja unser Sektor befindet, genauer an.

| DAM | Sektor | CRC |
|---|---|---|
| $FB | 512 Datenbyte | $F7 |

Das Daten-Feld beginnt mit einem DATA-ADDRESS-MARK, welches den Start des Sektors kennzeichnet. Der Wert $FB wird von einem späteren READ-SECTOR-Kommando als ein 'normales' DATA-ADDRESS-MARK interpretiert, wogegen der Wert $F8, der statt $FB eingetragen werden kann, als ein 'gelöschtes' DAM betrachtet wird.

Das Sektor-Feld wird beim Formatieren mit 'Dummy-Bytes' gefüllt. Die Werte können zwar frei bestimmt werden, sollten aber keinesfalls größer als $F4 sein. Die Anzahl kann 128, 256, 512 oder 1024 betragen. Wie der FDC die unterschiedlichen Sektor-Längen erkennt, wird im "ID-Feld" erklärt.

Durch die Übergabe des Wertes '$F7' veranlaßt, schreibt der FDC den Inhalt seines 16-Bit-CRC-Registers, welches eine Prüfsumme enthält, auf die Diskette. Obwohl nur ein Byte übergeben wird, werden vom Controller zwei Byte geschrieben. Die

Gesamtlänge des Datenfeldes beträgt bei einer Sektorgröße von 512 Bytes, wie auch in unserem Beispiel, 515 Bytes.

**Das Index-Feld**

Das Index-Feld oder kürzer ID-Feld enthält Informationen über das nachfolgende Datenfeld.

| ID-AM | Spur | Seite | Sektor | Länge | CRC |
|---|---|---|---|---|---|
| $FE | 00-79 | 00-01 | 00-09 | 00-03 | $F7 |

Das Index-Address-Mark (ID-AM) ist die Startmarkierung des ID-Feldes. Trifft der Controller, bei späteren Lese-Operationen, auf ein ID-AM, so wird er die 6 folgenden Bytes, bei ausgeschaltetem Address-Mark-Detector lesen. Das ID-AM wird ebenfalls ohne Takt-Impulse geschrieben.

Die drei Byte, die sich dem ID-AM anschließen, beschreiben die Lage des RECORDS. Hier wird zuerst die Spur-Nummer, auf der er sich befindet, angegeben. In unserem Fall ein Wert zwischen 0 und 79, abhängig davon, welche Spur formatiert wird.

Das Seiten-Feld gibt an, ob sich der Record auf der Vorder- oder auf der Rückseite befindet. Dieses Byte wird, vom Controller selbst, in keiner Weise bei irgendwelchen Operationen benutzt. Das Sektor-Feld schließlich enthält die Nummer des Sektors (1-9). Da der FDC unterschiedliche Sektorgrößen unterstützt, muß ihm mitgeteilt werden, wieviele Daten-Bytes im folgenden Sektor enthalten sind. Dies geschieht durch die Angabe im Längen-Feld.

Tabelle der Sektorlängen

| Längen-Feld | Byte pro Sektor |
|---|---|
| 00 | 128 |
| 01 | 256 |
| 02 | 512 |
| 03 | 1024 |

Für eine Sektorgröße von 512 Bytes steht in diesem Feld also eine '02'. Es fehlt nun nur noch die Prüfsumme. Das Schreiben dieser Summe wird, genau wie im Daten-Feld, durch Übergabe des Wertes '$F7' erreicht.

In der Addition ergibt sich, daß die Länge eines ID-Feldes immer 7 Bytes beträgt. Nachdem jetzt alle Komponenten und deren Anordnung innerhalb der Spur erklärt wurden, sind wir in der Lage die Record-Länge zu berechnen.

```
Daten-Feld      515 Bytes
ID-Feld     +     7 Bytes
GAP2-GAP4   +    92 Bytes

Record-Länge =  614 Bytes
```

Dies ist die tatsächliche Größe des Records. In unserem Puffer beträgt dessen Länge nur 612 Bytes, da für das Schreiben der beiden Prüfsummem (4 Bytes), jeweils nur ein Byte an den FDC übergeben wird.

In unserer Berechnung ist jedoch der Platz entscheidend, den ein Record auf der Spur einnimmt, also 614 Bytes. Für 9 Records benötigen wir also 9 * 614 Bytes = 5526 Bytes. Subtrahieren wir diese von den 6190 zur Verfügung stehenden Byte, so bleiben noch 664 Bytes für den Spur-Nachspann (GAP5) übrig.

Das ist mehr als ausreichend, wenn man bedenkt, daß hierfür nur 16 Bytes erforderlich sind. Selbst ein Format, welches 10 Sektoren a' 512 Bytes benutzt, würde die Länge von GAP5, mit 50 Bytes nicht unter das Mindestmaß sinken lassen.

Sehen wir uns zum Schluß an, mit welchen Daten unser Puffer aufbereitet wird. Zu der Tabelle, die wir hierfür angelegt haben, sind noch einige Erläuterungen nötig:

Die Daten von GAP2 bis einschließlich GAP4 (ein vollständiger Record) wiederholen sich für jeden Sektor. So muß z.B bei dem Format mit 29 Sektoren, der entsprechende Block, 29 mal hintereinander in den Puffer geschrieben werden. Die mit '$XX' angegebenen Werte müssen von Ihnen selbst bestimmt werden, was aber nicht sonderlich kompliziert ist. Wird z.B die Spur 54 formatiert, so wird als Wert für die Spur-Nr. jeweils eine '54' eingetragen.

Ein Wert für die Seiten-Nr. ist im allgemeinen '0' für die Vorderseite und '1' für die Rückseite.

Die Sektor-Nr. wird fortlaufend, normalerweise mit '1' beginnend, vergeben. Die Reihenfolge ist variabel und könnte ,z.B. bei einem Format mit 9 Sektoren, 3,6,9,1,4,7,2,5,8 lauten. Wichtig ist nur, daß die Folge vollständig ist.

Werden die Sektoren z.B. mit 1,2,3,5,6,7,8,9,10 bezeichnet, so wird ein späterer READ- SECTOR- bzw. WRITE-SECTOR-Befehl mit gesetztem m-Bit nach dem 3. Sektor mit RECORD-NOT-FOUND-Error abgebrochen, da der FDC keinen Sektor mit der Nr.4 finden kann.

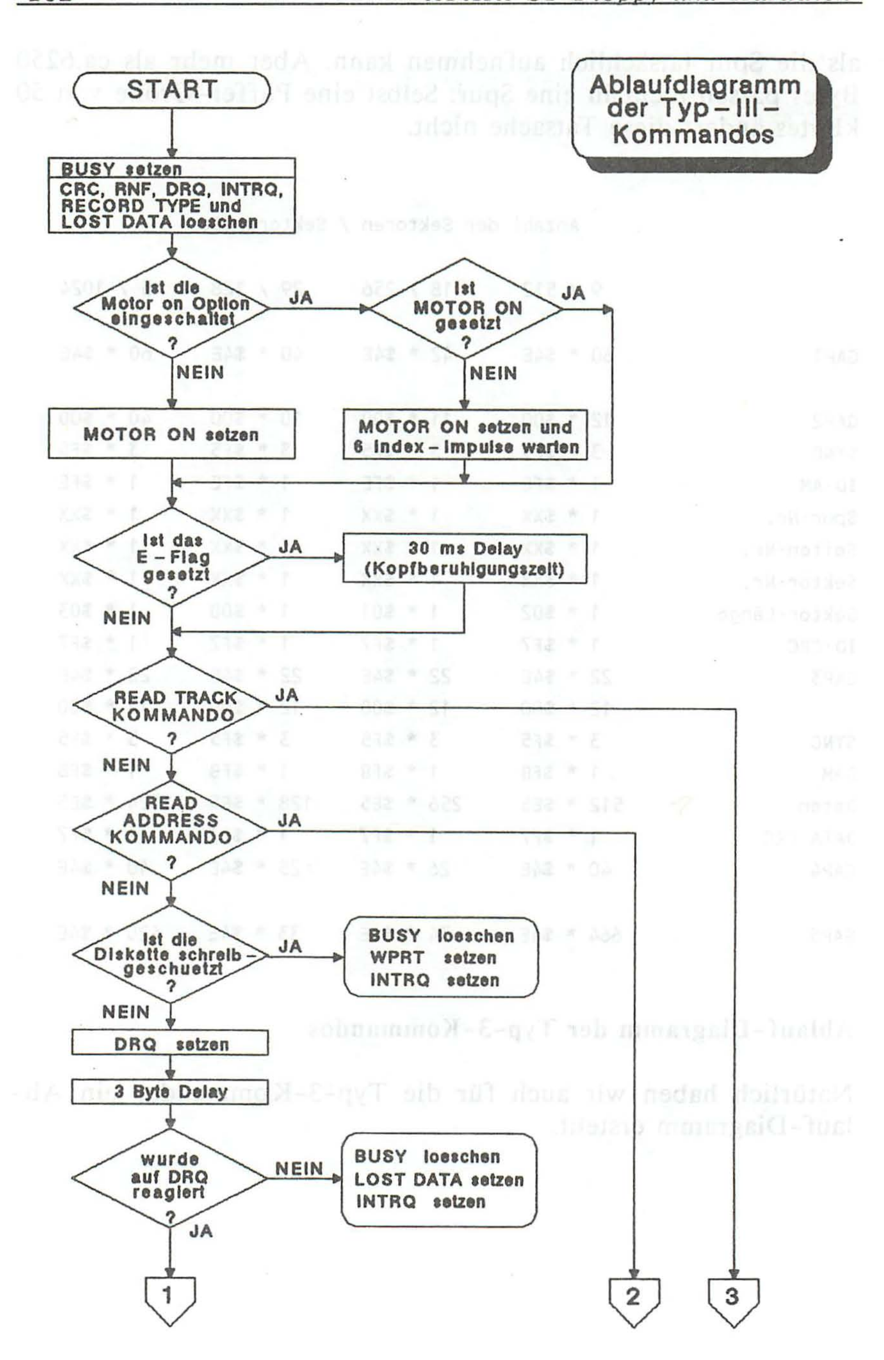
Das ATARI-FORMAT ist in der erste Spalte aufgeführt. Wer einen Blick in 'A Hitchhiker's Guide to the BIOS' wirft, wird in einem Punkt eine Abweichung zu unserer Tabelle entdecken. Die Länge des Spur-Nachspanns (GAP5) ist nicht mit 664 Bytes, sondern mit 1401 Bytes angegeben. Wenn man alle dort angegebenen Bytes addiert, so hat es den Anschein, daß die Spur-Länge ca. 7000 Bytes beträgt, was aber nicht der Fall ist. Vielmehr wird nur ein Puffer aufbereitet, der eben etwas größer ist

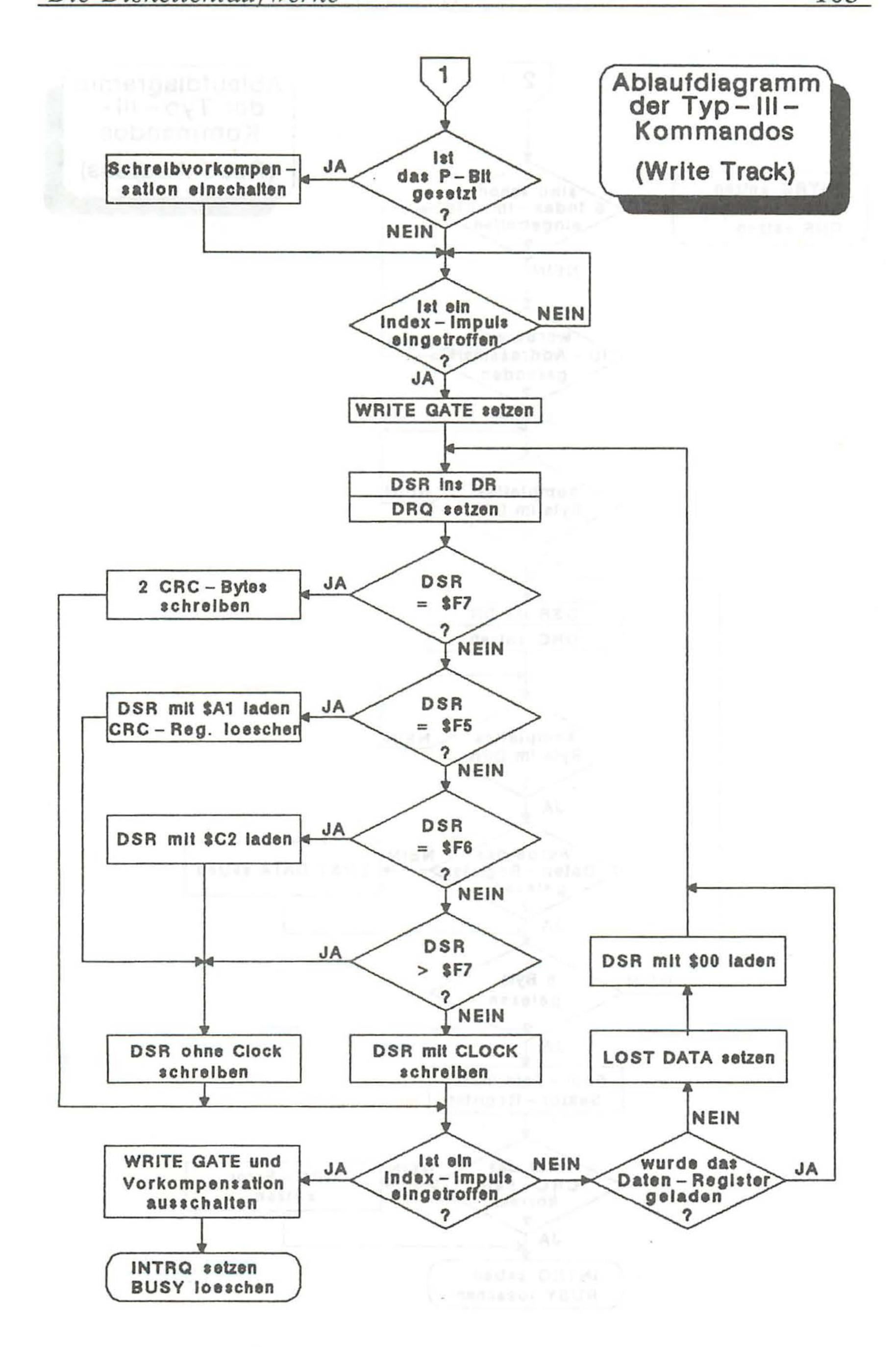
als die Spur tatsächlich aufnehmen kann. Aber mehr als ca.6250 Bytes passen nicht in eine Spur. Selbst eine Puffer-Größe von 50 kBytes ändert diese Tatsache nicht.

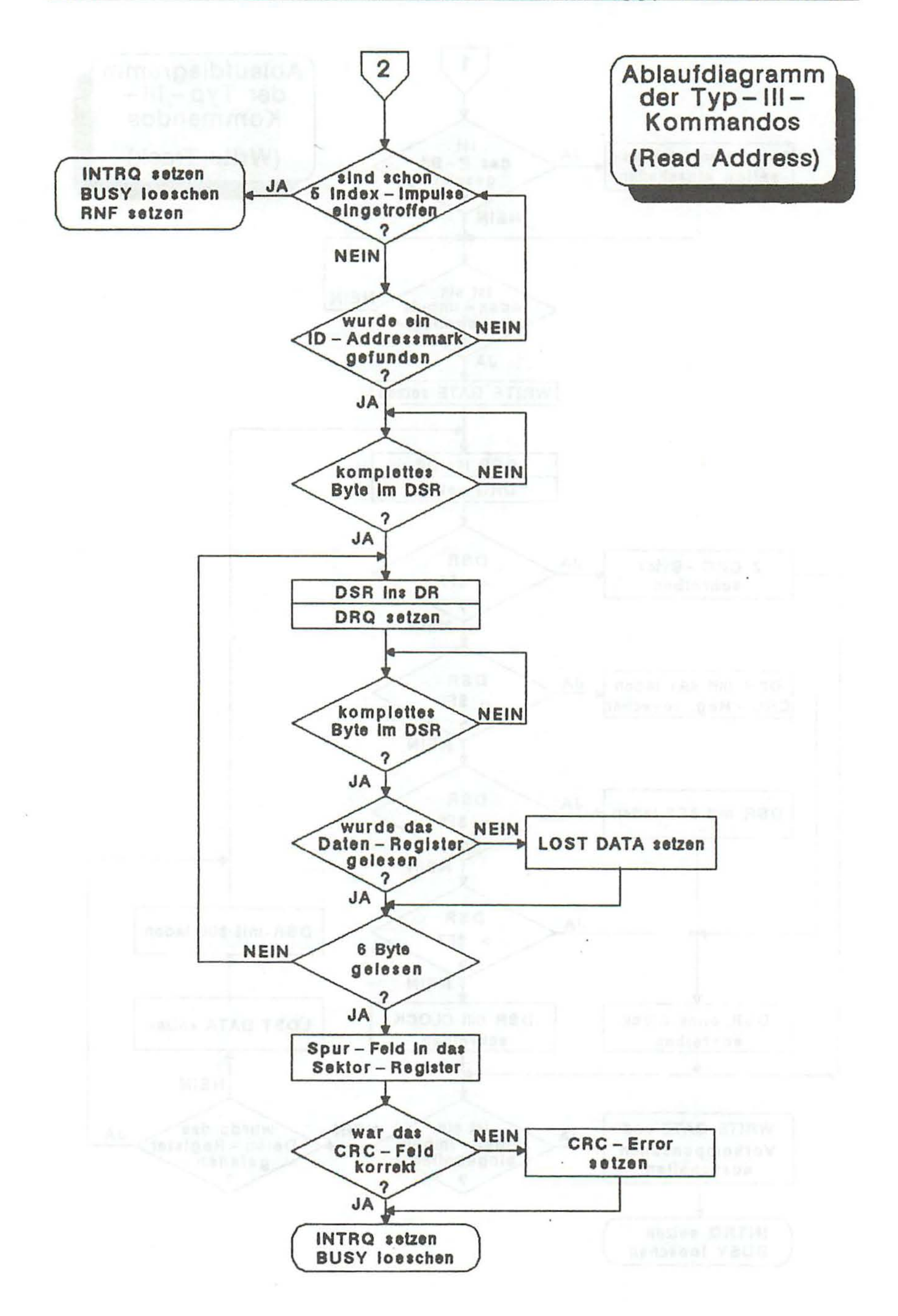
| | Anzahl der Sektoren / Sektorgröße | | | |
|---|---|---|---|---|
| | 9 / 512 | 18 / 256 | 29 / 128 | 5 / 1024 |
| GAP1 | 60 * $4E | 42 * $4E | 40 * $4E | 60 * $4E |
| GAP2 | 12 * $00 | 11 * $00 | 10 * $00 | 40 * $00 |
| SYNC | 3 * $F5 | 3 * $F5 | 3 * $F5 | 3 * $F5 |
| ID-AM | 1 * $FE | 1 * $FE | 1 * $FE | 1 * $FE |
| Spur-Nr. | 1 * $XX | 1 * $XX | 1 * $XX | 1 * $XX |
| Seiten-Nr. | 1 * $XX | 1 * $XX | 1 * $XX | 1 * $XX |
| Sektor-Nr. | 1 * $XX | 1 * $XX | 1 * $XX | 1 * $XX |
| Sektor-Länge | 1 * $02 | 1 * $01 | 1 * $00 | 1 * $03 |
| ID-CRC | 1 * $F7 | 1 * $F7 | 1 * $F7 | 1 * $F7 |
| GAP3 | 22 * $4E | 22 * $4E | 22 * $4E | 22 * $4E |
| | 12 * $00 | 12 * $00 | 12 * $00 | 12 * $00 |
| SYNC | 3 * $F5 | 3 * $F5 | 3 * $F5 | 3 * $F5 |
| DAM | 1 * $FB | 1 * $FB | 1 * $FB | 1 * $FB |
| Daten | 512 * $E5 | 256 * $E5 | 128 * $E5 | 1024 * $E5 |
| DATA-CRC | 1 * $F7 | 1 * $F7 | 1 * $F7 | 1 * $F7 |
| GAP4 | 40 * $4E | 26 * $4E | 25 * $4E | 40 * $4E |
| GAP5 | 664 * $4E | 34 * $4E | 33 * $4E | 420 * $4E |

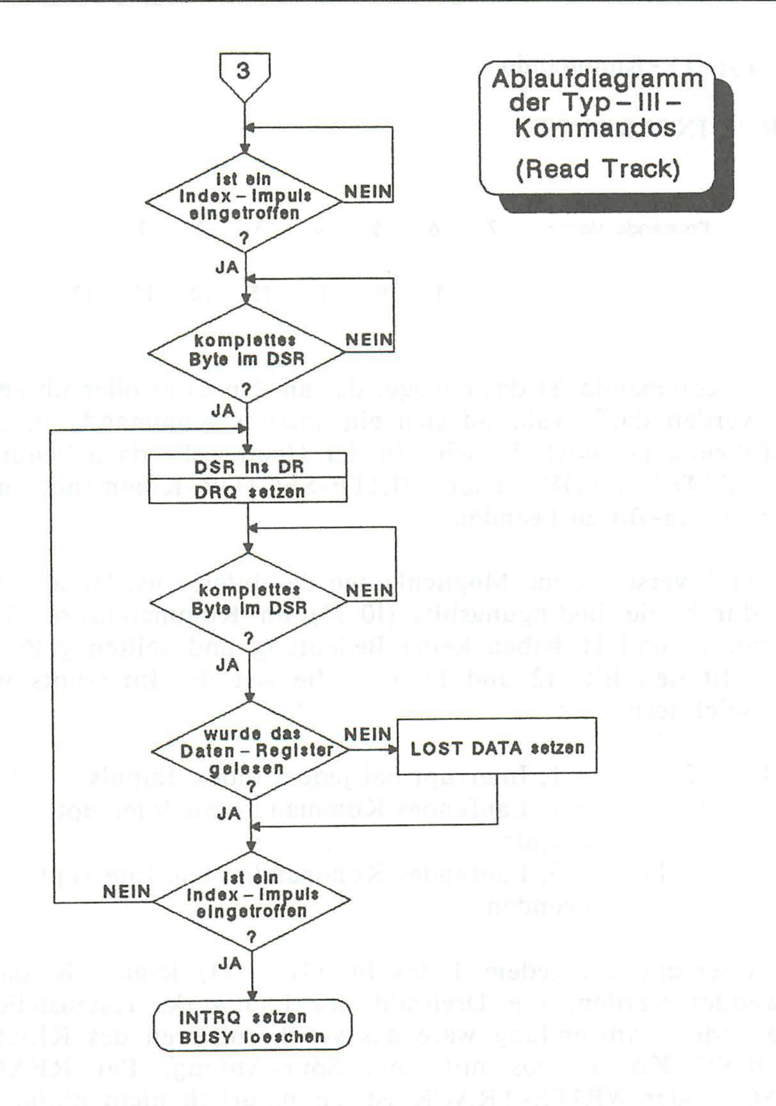
### Ablauf-Diagramm der Typ-3-Kommandos

Natürlich haben wir auch für die Typ-3-Kommandos ein Ablauf-Diagramm erstellt.


Ablaufdiagramm der Typ-III-Kommandos
START
BUSY setzen
CRC, RNF, DRQ, INTRQ, RECORD TYPE und LOST DATA loeschen
Ist die Motor on Option eingeschaltet ?
JA
NEIN
Ist MOTOR ON gesetzt ?
JA
NEIN
MOTOR ON setzen
MOTOR ON setzen und 6 Index-Impulse warten
Ist das E-Flag gesetzt ?
JA
NEIN
30 ms Delay (Kopfberuhigungszeit)
READ TRACK KOMMANDO ?
JA
NEIN
READ ADDRESS KOMMANDO ?
JA
NEIN
Ist die Diskette schreib-geschuetzt ?
JA
NEIN
BUSY loeschen
WPRT setzen
INTRQ setzen
DRQ setzen
3 Byte Delay
wurde auf DRQ reagiert ?
NEIN
JA
BUSY loeschen
LOST DATA setzen
INTRQ setzen
1
2
3


Ablaufdiagramm der Typ-III-Kommandos (Write Track)
1
Ist das P-Bit gesetzt ?
JA
Schreibvorkompen-sation einschalten
NEIN
Ist ein Index-Impuls eingetroffen ?
NEIN
JA
WRITE GATE setzen
DSR ins DR
DRQ setzen
DSR = $F7 ?
JA
2 CRC-Bytes schreiben
NEIN
DSR = $F5 ?
JA
DSR mit $A1 laden CRC-Reg. loeschen
NEIN
DSR = $F6 ?
JA
DSR mit $C2 laden
NEIN
DSR > $F7 ?
JA
NEIN
DSR ohne Clock schreiben
DSR mit CLOCK schreiben
Ist ein Index-Impuls eingetroffen ?
JA
WRITE GATE und Vorkompensation ausschalten
INTRQ setzen BUSY loeschen
NEIN
wurde das Daten-Register geladen ?
JA
NEIN
LOST DATA setzen
DSR mit $00 laden


2
Ablaufdiagramm der Typ-III-Kommandos (Read Address)
INTRQ setzen
BUSY loeschen
RNF setzen
JA
sind schon 5 Index-Impulse eingetroffen ?
NEIN
wurde ein ID-Addressmark gefunden ?
NEIN
JA
komplettes Byte im DSR ?
NEIN
JA
DSR ins DR
DRQ setzen
komplettes Byte im DSR ?
NEIN
JA
wurde das Daten-Register gelesen ?
NEIN
LOST DATA setzen
JA
6 Byte gelesen ?
NEIN
JA
Spur-Feld in das Sektor-Register
war das CRC-Feld korrekt ?
NEIN
CRC-Error setzen
JA
INTRQ setzen
BUSY loeschen


3
Ablaufdiagramm der Typ-III-Kommandos
(Read Track)
Ist ein Index-Impuls eingetroffen ?
NEIN
JA
komplettes Byte im DSR ?
NEIN
JA
DSR ins DR
DRQ setzen
komplettes Byte im DSR ?
NEIN
JA
wurde das Daten-Register gelesen ?
NEIN
LOST DATA setzen
JA
Ist ein Index-Impuls eingetroffen ?
NEIN
JA
INTRQ setzen
BUSY loeschen

**Das Typ-IV-Kommando**

**FORCE INTERRUPT**

| Kommando-Wort: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | 1 | 1 | 0 | 1 | I3 | I2 | I1 | I0 |

Dieses Kommando ist das einzige, das an den Controller übergeben werden darf, während sich ein anderes Kommando in der Ausführung befindet. Es wird in der Hauptsache dazu benutzt, ein READ-SECTOR- oder WRITE-SECTOR-Kommando mit gesetztem m-Bit zu beenden.

Es gibt 3 verschiedene Möglichkeiten des Interrupts. Diese werden durch die Bedingungsbits (I0-I3) im Kommandowort bestimmt. I0 und I1 haben keine Bedeutung und sollten gelöscht sein. Mit den Bits I2 und I3 wird die Art des Interrupts wie folgt selektiert:

| | | |
|---|---|---|
| ($D4) | I2 | = 1, Interrupt bei jedem Index-Impuls |
| ($D8) | I3 | = 1, Laufendes Kommando mit Interrupt beenden |
| ($D0) | I2-I3 | = 0, Laufendes Kommando ohne Interrupt beenden |

Der Interrupt bei jedem Index-Impuls ($D4) kann z.B. dazu verwendet werden, die Drehzahl des Laufwerks festzustellen. Eine andere Anwendung wäre das synchronisieren des READ-ADDRESS-Kommandos mit dem Spur-Anfang. Bei READ-TRACK oder WRITE-TRACK ist das natürlich nicht nötig, da sie ohnehin erst mit dem Index-Impuls gestartet werden.

Durch die Interrupts '$D8' und '$D0' kann ein in der Ausführung befindliches Kommando abgebrochen werden. Es ist zu beachten, daß der INTRQ-Ausgang nach einem '$D8'-Interrupt nicht, wie üblich, durch Lesen oder Schreiben des Kommando-Registers zurückgesetzt wird. Dies kann nur erreicht werden,

wenn dem '$D8'- ein '$D0'-Interrupt folgt und danach das Status-Register gelesen wird.

Wird dem FDC ein FORCE-INTERRUPT-Kommando übergeben, so muß vor dem nächsten Befehl eine Wartezeit von 16 Mikrosekunden eingelegt werden, da sonst das Interrupt-Kommando nicht ausgeführt wird.

#### 4.2.1.4 Status-Interpretation

Es hat sich herausgestellt, daß das Programmieren von Peripheriebausteinen, die für einen Daten-Transfer zuständig sind, fast ausschließlich dem Betriebssystem überlassen wird. Allerdings bieten diese Bausteine - in den meisten Fällen - erheblich mehr 'Features', als für den normalen Systembetrieb erforderlich sind. Dies ist der Grund dafür, daß die weniger gebräuchlichen Fähigkeiten nicht vom Betriebssystem ausgenutzt werden.

In Einzelfällen können diese 'schlummernden' Talente der Peripherie-Chips aber ein Programmierproblem drastisch vereinfachen oder gar erst lösen. Warum trauen sich also nur - vergleichsweise wenige - Programmierer, diese 'Talente' zu wekken? Nur etwa, weil es keinen passenden Betriebssystem-Aufruf dafür gibt? Nein - die Scheu eine solche Programmierung selbst vorzunehmen liegt meist in einer 'Angst vor dem Status' begründet.

Das äußert sich dann so, daß man zwar weiß wie man den Chip dazu veranlaßt eine bestimmte Aufgabe zu erledigen, aber den Status, den man zurückerhält, nicht interpretieren kann.

Oft ist nicht bekannt, welchen Status man nach einer fehlerfreien Ausführung erhält, denn irgendwelche Bits sind im Statusregister fast immer gesetzt. Ein O.K.-Status kann also durchaus variieren.

Wer schon hier nicht weiter weiß - wie soll er dann erst nach Erhalt eines Fehler-Status, den Programmteil der dem Aufruf folgt, weiterkodieren?

Es ist zeitraubend, alle Möglichkeiten durchzuspielen und somit experimentell zu ermitteln, in welchen Fällen man welchen Status erhält. Aber das muß auch nicht sein. Diese Aufgabe haben wir - jedenfalls was den Floppy-Controller betrifft - schon für Sie erledigt.

Wir zeigen an dieser Stelle nochmals die Bedeutung der einzelnen Status-Bits, die bereits in Kapitel 4.2.1.2 eingehend beschrieben wurden:

Status-Register des FDC

| Bit | Benennung | Bit=1 bedeutet |
|---|---|---|
| 7 | Motor On (MO) | Motor läuft |
| 6 | Write Protect (WPRT) | Disk schreibgeschützt |
| 5 | Record Type / Spin Up | DATA-MARK gelöscht |
| | bzw. Spin Up | Drehzahl erreicht |
| 4 | Record not found (RNF) | Sektor nicht gefunden |
| 3 | CRC-Error (CRC) | Prüfsummen-Fehler |
| 2 | Spur 0 | Kopf auf Track 0 |
| | bzw. Lost Data | Datenverlust |
| 1 | Index-Impuls | Index-Puls-Status |
| | bzw. Data Request | Übertragungsbereit |
| 0 | Busy | Kommando aktiv |

Es ist wissenswert, daß schon nach einer Positionierung des Schreib/Lese-Kopfes festgestellt werden kann, ob eine schreibgeschützte Diskette (oder keine) in das Laufwerk eingelegt wurde.

Gemeinsam für alle Kommandos (Typ-1 bis Typ-3) gilt, daß im Statuswort, welches direkt im Anschluß nach einem Kommando gelesen wird, Bit-7 gesetzt (der Motor wird nicht sofort ausge –

schaltet) und Bit-0 gelöscht ist (der FDC hat das Kommando ja beendet). Nach einem Typ-1-Kommando ist außerdem noch Bit-5 (Spin-Up) gesetzt.

### Der Status nach einem Typ-1-Kommando

Beginnen wir mit dem korrekten Status nach einem Typ-1-Kommando. In den verwendeten Kommandoworten sind die 'Stepping-Rate-Bits' für 3ms 'Track to Track' gesetzt (r0=1, r1=0).

| RESTORE-Kommando | normal | schreibgeschützt |
|---|---|---|
| 01 (mit MO-Option, ohne Verify) | A4 | E4 |
| 01 ( siehe (1) ) | A6 | E6 |
| 05 (mit MO-Option, mit Verify) | A4 | E4 |
| 09 (ohne MO-Option, ohne Verify) | A4 | E4 |
| 0D (ohne MO-Option, mit Verify) | A4 | E4 |

| SEEK-Kommando | normal | schreibgeschützt |
|---|---|---|
| 11 (mit MO-Option, ohne Verify) | A0 | E0 |
| 11 ( siehe (3) ) | A2 | E2 |
| 11 ( siehe (2) ) | A4 | E4 |
| 11 ( siehe (1) ) | A6 | E6 |
| 15 (mit MO-Option, mit Verify) | A0 | E0 |
| 15 ( siehe (2) ) | A4 | E4 |
| 19 (ohne MO-Option, ohne Verify) | A0 | E0 |
| 19 ( siehe (2) ) | A4 | E4 |
| 1D (ohne MO-Option, mit Verify) | A0 | E0 |
| 1D ( siehe (2) ) | A4 | E4 |

| STEP, STEP-IN, STEP-OUT | normal | schreibgeschützt |
|---|---|---|
| x1 (mit MO-Option, ohne Verify) | A0 | E0 |
| x1 ( siehe (2) ) | A4 | E4 |
| x5 (mit MO-Option, mit Verify) | A0 | E0 |
| x5 ( siehe (2) ) | A4 | E4 |
| x9 (ohne MO-Option, ohne Verify) | A0 | E0 |
| x9 ( siehe (2) ) | A4 | E4 |
| xD (ohne MO-Option, mit Verify) | A0 | E0 |
| xD ( siehe (2) ) | A4 | E4 |

(1) Dieser Wert gilt, wenn sich der Schreib/Lese-Kopf schon vor dem RESTORE- bzw. einem SEEK-Kommando nach Spur 0, über der Spur 0 befand. Es ist, neben dem Spur-0-Bit, das IP-Bit gesetzt. Das liegt daran, daß bei der Motor-On-Option, 6 Index-Impulse abgewartet werden. D.h., der FDC stellt während eines Index-Impulses fest, daß die gewünschte Spur erreicht ist, und beendet das Kommando.

(2) Diesen Status trifft man nach einem SEEK-, STEP- oder STEP-OUT-Kommando an, wenn der Schreib/Lese-Kopf über Spur 0 positioniert wird.

(3) Befindet sich der Schreib/Lese-Kopf bei einem SEEK-Kommando bereits über der gewünschten Spur (außer Spur 0), so ist im Status-Wort das IP-Bit gesetzt. Es liegt hier im Prinzip der gleiche Sachverhalt wie unter (1) beschrieben zugrunde.

Ein Fehler-Status nach einem Typ-1-Kommando kann im allgemeinen nur erhalten werden, wenn im Kommandowort das Verify-Bit gesetzt war. Im Statuswort ist dann noch zusätzlich:

a. falls kein ID-Feld gefunden wurde, das RNF-Bit gesetzt

und

b. falls kein korrektes ID-Feld gefunden wurde, das RNF- und das CRC-Bit gesetzt.

Somit ergibt sich ein Status von 'B2' oder 'BA' bzw.'F2' oder 'FA' bei einer schreibgeschützten Diskette. Geschieht das ganze auf Spur 0, so ist natürlich auch noch das Spur-0-Bit gesetzt. Das Statuswort hat dann den Wert 'B6' oder 'BE' bzw. 'F6' oder 'FE' bei einer schreibgeschützten Diskette.

Es fällt auf, daß im Fehlerfall immer das IP-Bit gesetzt ist. Das hat hier aber nichts mit der Motor-On-Option zu tun, sondern erklärt sich dadurch, daß die (vergebliche) Suche nach einem ID-Feld mit dem sechsten Index-Impuls abgebrochen wird.

**Der Status nach einem Typ-2-Kommando**

Bei den Typ-2-Kommandos gestaltet sich die Status-Interpretation etwas einfacher.

Nach einem erfolgreichen WRITE-SECTOR-Kommando enthält das Status-Register immer den Wert '80'. Nach einem READ-SECTOR-Kommando kann das Statuswort, falls ein Sektor mit 'gelöschtem' Data-Mark gelesen wurde, auch 'A0' betragen. Ansonsten ist hier ebenfalls eine '80' zu finden.

Wenn das Kommando nicht erfolgreich war, so ist der Status nach einem WRITE-SECTOR-Kommando entweder:

a. 'C0', nach dem Versuch eine schreibgeschützte Diskette zu beschreiben,

b. '90', wenn das zum gewünschten Sektor gehörende ID-Feld nicht gefunden wurde,

c. '88', falls die Prüfsumme (CRC) des ID-Feldes nicht korrekt war

oder

d. '84', wenn nicht auf ein 'DATA-REQUEST' des FDC reagiert wurde.

Nach einem fehlerhaften READ-SEKTOR-Kommando:

a. '90', wenn das zum gewünschten Sektor gehörende ID-Feld oder das DATA-MARK nicht gefunden wurde,

b. '98', falls die Prüfsumme (CRC) des ID-Feldes nicht in Ordnung war,

c. '88', wenn die Prüfsumme (CRC) im Daten-Feld einen Fehler aufwies

oder

d. '84', wenn nicht auf ein 'DATA-REQUEST' des FDC reagiert wurde.

**Der Status nach einem Typ-3-Kommando**

Noch einfacher ist der Status nach einem Typ-3-Kommando auszuwerten. Der Wert '80' zeigt auch hier eine erfolgreiche Ausführung an.

Für den Fehlerfall nach einem WRITE-TRACK-Kommando ergibt sich:

a. 'C0', bei einer schreibgeschützten Diskette

oder

b. '84', es wurde nicht auf einen DRQ des FDC reagiert.

Eine fehlerhafte Ausführung des READ-TRACK-Kommandos gibt es eigentlich gar nicht. Der FDC liest einfach nur, zwischen zwei Index-Impulsen, den RD-Eingang. Ganz gleich, ob eine Diskette im Laufwerk ist oder nicht.

Der einzige Fehler der vorstellbar ist, nämlich ein LOST-DATA (Status '84'), kann aufgrund eines Softwarefehlers in einem Mikroprogramm des FDC, nicht ausgewertet werden. Das LOST-DATA-Bit wird, außer bei einem Datenverlust, auch noch abhängig vom gelesenen Format gesetzt.

Es kann also nach einem READ-TRACK-Kommando nicht festgestellt werden, ob ein gesetztes LOST-DATA-Bit tatsächlich einen Datenverlust anzeigt.

Es bleibt noch das READ-ADDRESS-Kommando übrig. Auch hier ist ein Status von '80' als gut zu werten. Ansonsten könnte sich noch ergeben:

(a) '90', wenn kein ID-Feld gefunden wurde,

(b) '88', falls der FDC einen Prüfsumme-Fehler im ID-Feld erkannt hat

(c) '84', wenn nicht auf eine Datenanforderung reagiert wurde.

### 4.2.2 Die Floppy-Schnittstelle

Der etwas merkwürdige Stecker an der Rückseite des ST hat 14 Pole. Mit diesen 14 Leitungen wird die gesamte Steuerung der Laufwerke und die Datenübertragung abgewickelt. Der Ablauf dieser Steuerung ist recht einfach zu beschreiben, da die Diskettenlaufwerke keine Eigenintelligenz besitzen.

Dies hat einen großen Vorteil. Die Schnittstelle zu solchen Diskettenlaufwerken ist nämlich genormt. Es handelt sich dabei um eine sogenannte SHUGART-Schnittstelle, die sich an vielen

Laufwerken befindet. Nur deshalb ist es ja so einfach, Fremdlaufwerke an den ATARI ST anzuschließen.

Diese Schnittstelle besitzt einen 34-poligen Anschluß, der meistens für einen Flachbandkabel-Stecker vorgesehen ist. Die Hälfte dieser 34 Pins sind verbunden und führen den gemeinsamen Minuspol, also Masse. Bei einem Flachbandkabel liegen diese Massedrähte immer abwechselnd, da immer die ungeraden Steckerpins auf Masse liegen.

Dies hat den einfachen Grund, daß so zwischen zwei Signalleitungen immer eine Masseleitung liegt. Dadurch wird eine gewisse Abschirmung zwischen den Signalleitungen erreicht, die bei den recht hohen Taktraten der Signale wichtig ist.

Von den verbleibenden 18 Leitungen werden 14 mit dem ATARI verbunden. Betrachten wir nun diese Signale des SHUGART-Steckers:

**Pin 2** **Head Load**

Ein Null-Signal auf dieser Leitung bewirkt, daß der Schreib-/Lesekopf auf die Diskette aufgesetzt wird. Diese Maßnahme ist zur Schonung der Disketten vorgesehen, da der Kopf nur dann auf der Diskette schleift, wenn er wirklich zugreifen soll. Leider ist dieses Signal nicht aus dem ATARI ST herausgeführt, da der Floppy-Controller WD1772 nicht über diesen Anschluß verfügt. Oft wird jedoch diese Leitung mit 'Motor on' verbunden.

**Pin 3** **Masse**

Ab hier sind alle ungeraden Leitungen bis 33 an Masse angeschlossen. Dieses Minus wird sowohl für den Betrieb als auch zur Abschirmung verwendet.

**Pin 4** **in Use**

Dieses Signal soll dem Laufwerk anzeigen, daß es angeschlossen ist und benutzt wird. Auch dieser Anschluß wird nicht an den ATARI angeschlossen.

**Pin 6** **Drive Select 3**

Ein Null-Signal auf dieser Leitung bedeutet, daß das Laufwerk 3 angesprochen werden soll. Nur das Laufwerk, welches durch sogenannte Jumper, kleine Stecker im Laufwerk, als Laufwerk 3 ausgewiesen ist, reagiert im folgenden auf die weiteren Befehle; alle anderen verhalten sich neutral. Dieses Signal ist beim ATARI unbelegt, da maximal zwei Laufwerke angeschlossen werden können (0 und 1 bzw. A und B).

**Pin 8** **Index**

Auf dieser Leitung sendet das Laufwerk bei jeder Umdrehung der Diskette ein Null-Signal. Dieses Signal bedeutet für den Controller, daß die nun folgenden Daten ganz am Anfang des aktuellen Tracks stehen. Dadurch kann sich der Controller synchronisieren.

**Pin 10** **Drive Select 0**

Dieses Signal entspricht demjenigen auf Pin 6, nur daß hier Laufwerk 0 angesprochen wird (Disk A).

**Pin 12** **Drive Select 1**

Wie oben, nur Laufwerk 1 (Disk B).

**Pin 14 Drive Select 2**

Wie oben, nur Laufwerk 2. Nicht am ST angeschlossen, da nur zwei Laufwerke möglich sind.

**Pin 16 Motor on**

Ein 1-Signal an diesem Anschluß startet die Motoren aller angeschlossenen Laufwerke, eine Null stoppt sie wieder.

**Pin 18 Direction**

Dieses Signal gibt an, in welche Richtung der nächste Schritt des Schreib-/Lesekopfes gehen soll. Bei einer Null ist die Richtung nach innen, also zum Track 79 gewählt, eine Eins bedeutet nach außen, zum Track 0 hin.

**Pin 20 Step**

Ein Null-Impuls veranlaßt den Schrittmotor im Laufwerk, den Schreib-/Lesekopf einen Schritt in die durch 'Direction' angegebene Richtung zu bewegen.

**Pin 22 Write Data**

Diese Leitung führt die seriell übertragenen Daten, die auf die Diskette geschrieben werden sollen.

**Pin 24 Write Gate**

Dieses Signal wählt die Datenrichtung aus. Liegt hier eine Null, so wird auf die Diskette geschrieben, bei einer Eins wird gelesen. Ist der Schreibschutz auf der Diskette in der entsprechenden Position, so werden von dem Laufwerk selbst keine Schreibzugriffe zugelassen.

**Pin 26** **Track 0**

Befindet sich der Schreib-/Lesekopf über der Spur 0, so liegt hier ein Null-Signal an.

**Pin 28** **Write Protect**

Eine Null auf dieser Leitung bedeutet, daß die Diskette schreibgeschützt ist.

**Pin 30** **Read Data**

Über diese Leitung werden die gelesenen Daten zum Rechner geführt.

**Pin 32** **Side Select**

Über diese Leitung wird die gewünschte Seite der Diskette ausgewählt. Eine Null wählt die Seite 1 aus, eine Eins die Seite 0. Bei einseitigen Laufwerken ist diese Leitung unbenutzt.

**Pin 34** **Ready**

Eine Null auf dieser Leitung gibt an, daß eine Diskette im Laufwerk eingelegt ist und sich normal dreht. Mit Hilfe dieser Leitung kann der Rechner

feststellen, ob die Diskette gewechselt wird. Auch diese Leitung ist nicht am ATARI ST angeschlossen.

Alle diese Signale entsprechen in ihrem Pegel dem TTL-Standard, d. h. 0-0.4 Volt bedeutet LO (Null), 2.5-5.25 Volt bedeutet HI (Eins). Um diese Signale zu sichern, sind in den meisten Diskettenlaufwerken eine Reihe von 'Pull-up-Widerständen' eingebaut.

Werden mehrere Laufwerke parallel angeschlossen, empfiehlt es sich, diese Widerstände bis auf die des letzten (beim ATARI des zweiten) Laufwerkes herauszunehmen, da sonst die Ausgänge des ATARI überlastet werden. Bei einigen Laufwerkstypen (z.B. EPSON) sind die Widerstände zusammen als ein Bauteil gesteckt, so daß man sie leicht herausziehen kann. Bei den Original-Laufwerken von ATARI, übrigens auch EPSON-Laufwerke, ist das allerdings nicht erforderlich.

## 4.3 Anschluß der Diskettenlaufwerke

Die Diskettenlaufwerke, die von ATARI für den ST angeboten werden, sind recht einfach anzuschließen: Kabel einstecken und fertig.

Komplizierter wird es, wenn man ein anderes Laufwerk anschließen möchte. Das erste Problem, welches dabei auftritt, ist das des Anschlußsteckers, der nicht oder nur schwer erhältlich ist (Stand 5/86). Dabei kann man sich jedoch zur Not mit Lötnägeln behelfen, die man entweder auf eine eigens dafür angefertigte Platine auflötet oder nach Justage auf irgendeinem geeigneten Träger mit Kunstharz bzw. einem anderen Kunststoff eingießt.

Hat man sich also auf irgendeine Art einen passenden Stecker besorgt, so wird die Verdrahtung in Angriff genommen. Für die Leitung zum Laufwerk ist ein abgeschirmtes Kabel sehr empfehlenswert, wenn die Länge ca. 1 Meter übersteigt. Aufgrund der hohen Übertragungsraten treten nämlich elektrische Effekte

wie Induktivitäten und Kapazitäten auf, die die Datenübertragung ungünstig beeinflussen können. Am sichersten ist daher ein Kabel, in dem die einzelnen Adern gegeneinander abgeschirmt sind (z.B. SCART-Kabel).

Ist nun die Verbindung zwischen ATARI ST und Diskettenstation gesichert, so muß das Kabel noch angeschlossen werden. Hier noch einmal die Verdrahtungstabelle im Überblick:

| ATARI ST | Leitung | SHUGART-Stecker |
|---|---|---|
| 1 | Read Data | 30 |
| 2 | Side 0 select | 32 |
| 3 | Ground | alle ungeraden Anschlüsse |
| 4 | Index Pulse | 8 |
| 5 | Drive 0 select | 10 |
| 6 | Drive 1 select | 12 |
| 7 | Ground | s.o. |
| 8 | Motor on | 16 |
| 9 | Direction in | 18 |
| 10 | Step | 20 |
| 11 | Write Data | 22 |
| 12 | Write Gate | 24 |
| 13 | Track 00 | 26 |
| 14 | Write Protect | 28 |

Werden an diese Leitungen zwei Laufwerke angeschlossen, so werden alle Anschlüsse parallel geschaltet. Die Auswahl, welches der Laufwerke nun A bzw. B sein soll, geschieht direkt im Laufwerk. Dazu müssen sogenannte Jumper umgesteckt werden, kleine Stecker, die bestimmte Kontakte verbinden. Wo diese Jumper im Laufwerk liegen und wie sie zu stecken sind, entnehmen Sie dazu der Beschreibung des verwendeten Laufwerks.

| Eingabeprm. | Funkt. - Nr. | Laufw. - Nr. | Spur - Nr. | Start - Sektor - Nr. | Anzahl zu uebertr. Bytes | Anzahl zu lesender ID - Felder | Startadr. d. Spur - Puffers | Startadr. d. Sektor - Puffers | Startadr. d. ID - Feld Puffers | Startadr. d. ID - Status Puffers |
|---|---|---|---|---|---|---|---|---|---|---|
| FUNKTION | FDC%(12) | FDC%(13) | FDC%(14) | FDC%(15) | FDC%(16) | FDC%(17) | FDC%(26) FDC%(27) | FDC%(28) FDC%(29) | FDC%(30) FDC%(31) | FDC%(32) FDC%(33) |
| Restore | 00 | | | | | | | | | |
| Seek | 01 | | xx | | | | | | | |
| Step | 02 | | | | | | | | | |
| Step - In | 03 | | | | | | | | | |
| Step - Out | 04 | | | | | | | | | |
| Read - Sector | 05 | | | xx (1) | xx (2) | | | xxxx (4) | | |
| Write - Sektor | 06 | | | xx (1) | xx (2) | | | xxxx (4) | | |
| Read - Track | 07 | | | | xx (2) | | xxxx (4) | | | |
| Write - Track | 08 | | | | xx (2) | | xxxx (4) | | | |
| Read - Address | 09 | | | | | xx (3) | | | xxxx (4) | xxxx (4) |
| Force - Interrupt | 10 | | | | | | | | | |
| Lfw. selektieren | 11 | xx | | | | | | | | |
| Sektorreg. lesen | 12 | | | | | | | | | |
| Spurreg. lesen | 13 | | | | | | | | | |
| Statusreg. lesen | 14 | | | | | | | | | |
| Spurreg. schreiben | 15 | | xx | | | | | | | |

| Ausgabeprm. | Spur - Nr. | Sektor - Nr. | FDC - STATUS | DMA - STATUS | Timeout | DMA - Startadr. | DMA - Endadr. | Anzahl der uebertrag. Bytes |
|---|---|---|---|---|---|---|---|---|
| FUNKTION | FDC%(14) | FDC%(15) | FDC%(18) | FDC%(19) | FDC%(20) | FDC%(22) FDC%(23) | FDC%(24) FDC%(25) | FDC%(21) |
| Restore | | | xx | | xx | | | |
| Seek | | | xx | | xx | | | |
| Step | | | xx | | xx | | | |
| Step - In | | | xx | | xx | | | |
| Step - Out | | | xx | | xx | | | |
| Read - Sector | | | xx | xx | xx | xxxx | xxxx | xx |
| Write - Sector | | | xx | xx | xx | xxxx | xxxx | xx |
| Read - Track | | | xx | xx | xx | xxxx | xxxx | xx |
| Write - Track | | | xx | xx | xx | xxxx | xxxx | xx |
| Read - Address | | | xx | xx | xx | xxxx | xxxx | xx |
| Force - Interrupt | | | | | | | | |
| Lfw. selektieren | | | | | | | | |
| Sektorreg. lesen | | xx | | | | | | |
| Spurreg. lesen | xx | | | | | | | |
| Statusreg. lesen | | | xx | | | | | |
| Spurreg. schreiben | | | | | | | | |

# 5. Die Festplatte SH204

Nun kommen wir zu der etwas teureren, aber doch wesentlich schnelleren Art der Datenspeicherung: der Festplatte. Eine solche Festplatte, auch Harddisk genannt, wird auch für den ATARI ST angeboten, und zwar für einen recht günstigen Preis.

Was sind nun die Vor- und Nachteile einer Festplatte? Nun, der erste Nachteil liegt auf der Hand: eine Harddisk kostet wesentlich mehr als eine Diskettenstation. Außerdem ist es nicht möglich, durch Wechseln der Platte Programme auszutauschen oder eine Bibliothek anzulegen, wie es ja mit Disketten geht.

Doch betrachtet man die Vorteile einer Harddisk, so wird die Investition doch schmackhaft. Da wäre einmal die Geschwindigkeit, mit der der Datenaustausch zwischen ATARI ST und der Festplatte abläuft. Diese ist nämlich bis zu 10 mal höher als bei Diskettenoperationen.

Ein weiterer Vorteil ist die Kapazität einer Festplatte, die bei den momentan angebotenen Geräten 20 Megabyte beträgt. Auf eine solche Platte passen somit z.B. alle Programme und Dateien eines umfangreichen Compilers mitsamt der Quelldateien Ihrer C- oder PASCAL-Programme. Da diese Compiler meist diskettenorientiert arbeiten, d.h. ständig auf die Diskette (oder auch Harddisk) zugreifen müssen, ist dies schon ein großer Vorteil. Das ewige Wechseln der Disketten bei Verwendung nur eines Laufwerks entfällt dann.

Eine häufige Verwendung von Festplatten findet sich auch in der EDV, wo große Datenmengen verwaltet werden müssen. Dabei ist es unzumutbar, immer wieder Disketten zu wechseln. Stellen Sie sich einmal vor, die Angestellten einer Bank müßten für jeden Kontoauszug die entsprechende Diskette in den Bankrechner einlegen!

Nun, für die EDV einer Bank reicht ein ATARI ST mit einer 20 MByte Festplatte wohl nicht ganz aus. Wohl aber für die Verwaltung eines kleineren Betriebes, in dem die Lagerhaltung und

die Personalverwaltung mit einem Rechner bewältigt wird. Dies ist auch die Hauptanwendung von Harddisks.

Wir wollen uns nun einmal ansehen, wie eine solche Datenmenge von einer Festplatte und natürlich auch dem angeschlossenen Rechner bewältigt wird.

## 5.1 Funktion und Aufbau

Die Funktion einer Harddisk ist derjenigen der Disketten sehr ähnlich. Auch hier drehen sich eine oder mehrere Scheiben (in der ATARI-Harddisk nur eine) mit konstanter Geschwindigkeit und werden jeweils von einem Schreib-/Lesekopf überstrichen. Hier treten allerdings schon einige gravierende Unterschiede gegenüber Diskettenlaufwerken auf.

Die Rotationsgeschwindigkeit der Festplatte ist bedeutend höher als die der Diskette, um die hohe Geschwindigkeit des Datenzugriffs und der Datenübertragung zu ermöglichen.

Um nun den Schreib-/Lesekopf, der über einer so schnellen Scheibe (etwa 10mal schneller als eine Diskette) dahinfliegt, zu schonen, liegt er überhaupt nicht auf der Scheibe auf. Mit einem technischen Meisterstück wird nämlich erreicht, daß der Kopf in einem winzigen Abstand zur Platte bleibt. Dieser Spalt ist so winzig, daß sich dazu ein Staubkorn wie ein Felsbrocken ausmacht.

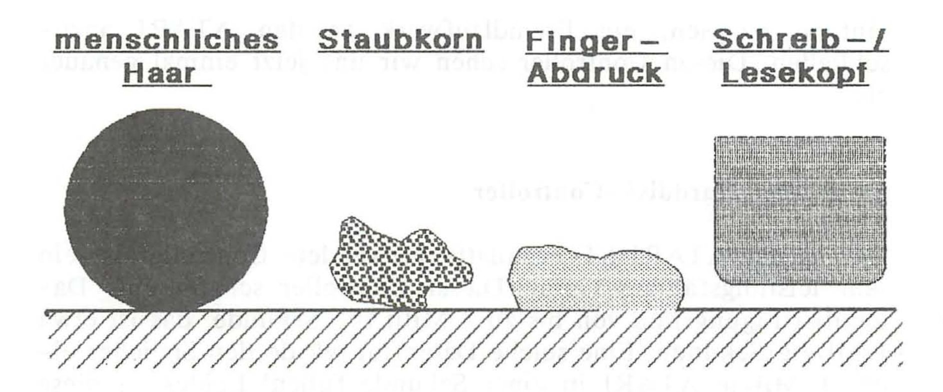



Bild: Vergleich Spalt, Staubkorn, Haar

So wie ein solches Staubkorn in obigen Bild wirkt, so wirkt es auch in der Praxis. Liegt eines auf der Platte und trifft mit der hohen Tangentialgeschwindigkeit der Platte auf den Kopf, so kann dies zu bösen Schäden an der Platte und/oder am Schreib-/Lesekopf führen. Einen solchen 'Unfall' nennt man 'Head-Crash'. Diese Vorfälle sind sehr gefürchtet, da sie meist teure Auswirkungen haben.

Um einen Head-Crash zu vermeiden, ist die Platte mitsamt dem Kopf in einem luftdichten Gehäuse verpackt. Dadurch erklärt sich auch, warum Harddisks nicht ebenso wie Disketten wechselbar sind. Auf dem Markt befinden sich zwar Wechselplatten-Laufwerke, die aber sehr aufwendig und dadurch auch sehr teuer sind. Außerdem wird momentan für den ATARI ST kein solches Laufwerk angeboten, so daß wir diese nicht weiter betrachten wollen.

Einen weiteren Unterschied zwischen einem Diskettenlaufwerk und der Harddisk am ATARI ST stellt der Controller dar. Der im ST eingebaute Floppy-Disk-Controller ist nur, wie der Name schon sagt, für die Diskettenlaufwerke zuständig. Die Harddisk dagegen hat ihren eigenen Controller, welcher auch im Gehäuse des Laufwerks eingebaut ist. Dadurch ist es leider nicht mehr so

einfach möglich, ein Fremdlaufwerk an den ATARI anzuschließen. Diesen Controller sehen wir uns jetzt einmal genauer an.

### 5.1.1 Der Harddisk-Controller

Der in der ATARI ST-Festplatte verwendete Controller ist ein sehr leistungsfähiges Gerät. Dieser Controller schafft eine Datenübertragungsrate von bis zu 8 MBit pro Sekunde, das ist etwa 1 MByte/Sekunde. Eine solche Datenflut würde den Speicher eines 1 MByte-ATARI in einer Sekunde füllen! Leider ist diese Zahl nicht für den wirklichen Datentransfer maßgeblich.

Eine starke 'Bremse' der Datenübertragung ist die Mechanik, die in der Festplatte steckt. Gemeint ist die Rotationsgeschwindigkeit der Platte und der Schrittmotor, der den Schreib-/Lesekopf erst einmal an die richtige Stelle, also über den richtigen Track fahren muß. Alle diese Punkte verringern die tatsächlich erreichbare Geschwindigkeit des Datenaustauschs, die dennoch sehr hoch ist.

Der Controller hat eine recht einfache innere Struktur. Dennoch ist sein Befehlssatz so vielseitig, daß er sogar Fehlerkorrektur unterstützt.

Die Hardware des Controllers besteht hauptsächlich aus einem Disk-Controller, einem Kodierer/Dekodierer und einem Mikrocontroller. Diese Einzelteile haben die folgenden Aufgaben bzw. Funktionen:

Der *Disk-Controller* wandelt die Daten vom seriellen ins parallele Format und umgekehrt. Außerdem wandelt er die Daten selbst in ein anderes Bitmuster um, welches dann wirklich auf die Platte geschrieben wird. Dieses andere Format ermöglicht durch einen Trick die Erkennung von einfachen Fehlern beim Lesen.

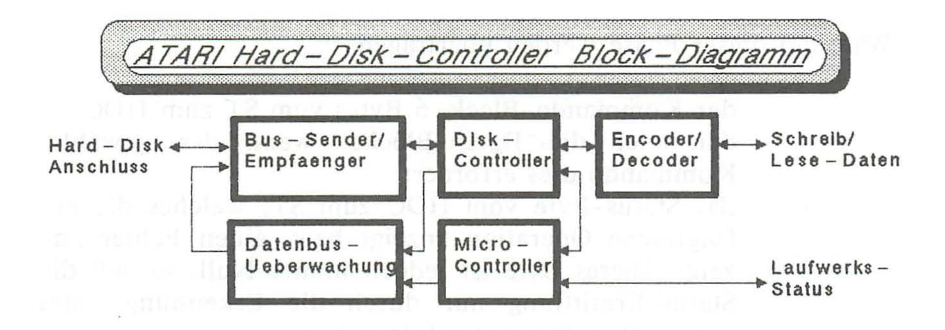
Der *Kodierer/Dekodierer* wandelt die vom Disk-Controller erhaltenen Daten in die elektrischen Signale um, die den Schreib-

kopf steuern. Umgekehrt wandelt er die Signale, die beim Lesen vom Kopf kommen, in Bits um, wobei er gleichzeitig als Datenseparator dient (s. Floppy-Disk-Controller).

Der Mikrocontroller arbeitet wie ein eigentlicher Disk-Controller. Seine Aufgaben sind:

- Interpretation der vom Rechner kommenden Kommandos
- Auswahl des angesprochenen Laufwerks (normalerweise ist nur eines vorhanden)
- Auswahl des Kopfes in dem Laufwerk (obere oder untere Plattenseite)
- Steuerung des Steppermotors, der den Schreib-/Lesekopf in die richtige Position fährt
- Statusermittlung

Hier nun ein einfaches Blockdiagramm des ATARI Hard-Disk-Controllers:




Die Operationen, die man über den DMA-Bus mit der Harddisk durchführt, sind in 5 verschiedene Phasen aufgeteilt. Diese Phasen sind folgendermaßen definiert:

**Reset-Phase**

tritt auf, wenn entweder die RESET-Taste am ST betätigt bzw. der Rechner eingeschaltet wird oder der RESET- Befehl der 68000-Maschinensprache auftritt. Der Bus und damit der HDC werden in den Grundzustand versetzt.

**Bus-frei-Phase**

liegt vor, wenn kein Gerät auf den Bus zugreift.

**Ziel-Anwahl-Phase**

beginnt durch den Aufruf eines Gerätes durch Rücksetzen der SEL-Leitung. Die Adressierung des gewünschten Gerätes geschieht durch ein gesetztes Datenbit der 8- Bit-Parallel-Leitung. Das adressierte Gerät (hier: HDC) antwortet mit einem BUSY-Signal, worauf die SEL-Leitung wieder gesetzt wird. Danach beginnt die

**Informations-Übertragungs-Phase**

Während dieser Phase werden übertragen:

- der Kommando-Block, 6 Bytes vom ST zum HDC
- der oder die Daten-Blöcke, wenn das gewählte Kommando dies erfordert
- das Status-Byte vom HDC zum ST, welches die erfolgreiche Operation anzeigt bzw. einen Fehler anzeigt. Dieses Byte ist jedoch immer Null, so daß die Status-Ermittlung nur durch die Erkennung eines eventuellen Timeout erfolgen kann.
- das Completion-Byte vom HDC zum ST, ein Null-Byte, welches das Ende der gesamten Operation signalisiert.

**Bus-Auslösungs-Phase**

wird durch Setzen der BUSY-Leitung ausgelöst und bedeutet, daß der Bus nun frei für die nächste Operation ist. Danach befindet sich der Bus wieder in der BUS-frei-Phase.

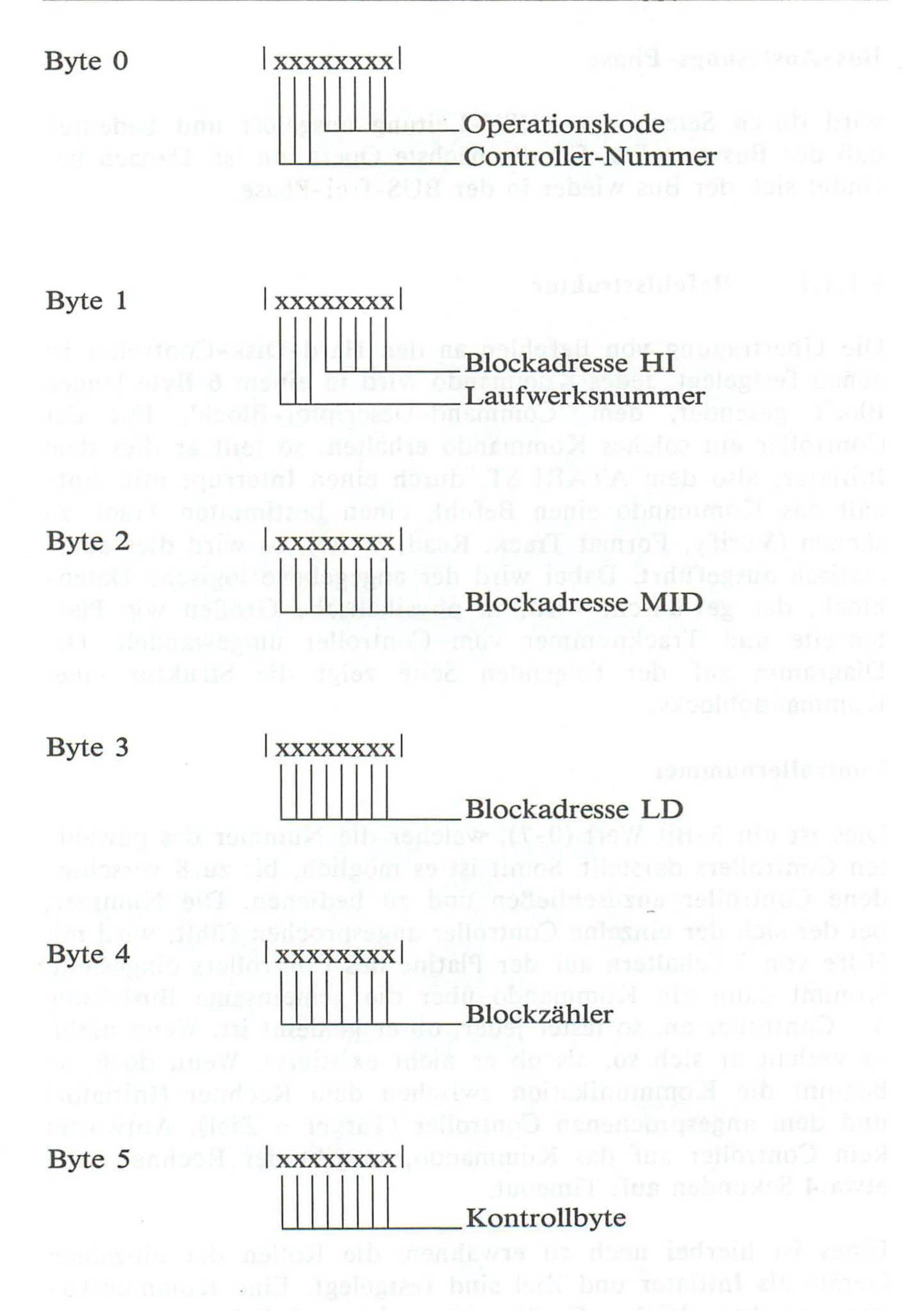
### 5.1.1.1 Befehlsstruktur

Die Übertragung von Befehlen an den Hard-Disk-Controller ist genau festgelegt. Jedes Kommando wird in einem 6 Byte langen Block gesendet, dem 'Command-Descriptor-Block'. Hat der Controller ein solches Kommando erhalten, so teilt er dies dem Initiator, also dem ATARI ST, durch einen Interrupt mit. Enthält das Kommando einen Befehl, einen bestimmten Track zu suchen (Verify, Format Track, Read, Write), so wird dies automatisch ausgeführt. Dabei wird der angegebene logische Datenblock, der gewünscht wird, in physikalische Größen wie Plattenseite und Tracknummer vom Controller umgewandelt. Das Diagramm auf der folgenden Seite zeigt die Struktur eines Kommandoblocks.

**Controllernummer**

Dies ist ein 3-Bit Wert (0-7), welcher die Nummer des gewählten Controllers darstellt. Somit ist es möglich, bis zu 8 verschiedene Controller anzuschließen und zu bedienen. Die Nummer, bei der sich der einzelne Controller angesprochen fühlt, wird mit Hilfe von 3 Schaltern auf der Platine des Controllers eingestellt. Kommt dann ein Kommando über die gemeinsame Busleitung der Controller an, so testet jeder, ob er gemeint ist. Wenn nicht, so verhält er sich so, als ob er nicht existierte. Wenn doch, so beginnt die Kommunikation zwischen dem Rechner (Initiator) und dem angesprochenen Controller (Target = Ziel). Antwortet kein Controller auf das Kommando, so gibt der Rechner nach etwa 4 Sekunden auf: Timeout.

Eines ist hierbei noch zu erwähnen: die Rollen der einzelnen Geräte als Initiator und Ziel sind festgelegt. Eine Kommunikation zwischen gleichen Geräten ist somit unmöglich.


Byte 0
xxxxxxxx
Operationskode
Controller-Nummer
Byte 1
xxxxxxxx
Blockadresse HI
Laufwerksnummer
Byte 2
xxxxxxxx
Blockadresse MID
Byte 3
xxxxxxxx
Blockadresse LD
Byte 4
xxxxxxxx
Blockzähler
Byte 5
xxxxxxxx
Kontrollbyte

**Operationskode**

Dieser Kode, auch OpCode genannt, enthält in 5 Bits das auszuführende Kommando. Dadurch sind nur Kommandos von 0 bis 31 möglich.

**Laufwerksnummer**

Ähnlich wie die Controllernummer ist dies eine 3-Bit-Zahl, die das gewählte Laufwerk bezeichnet. Jeder der bis zu 8 Controller kann somit jeweils bis zu 8 Laufwerke steuern, wodurch also theoretisch 64 Laufwerke an den ATARI anschließbar sind.

**Blockadresse**

Diese 21 Bit-Zahl bezeichnet den gewählten logischen Datensektor. Die Umrechnung dieser Zahl (bis zu 2097151) in die physikalischen Werte erledigt der Controller. Die ATARI-Festplatte enthält 41616 Sektoren, so daß die Blockadresse diesen Wert nicht überschreiten und, da die Blocknummer bei 0 beginnt, auch nicht erreichen.

**Blockzähler**

Dieser Zähler bestimmt die Anzahl der zu lesenden bzw. zu schreibenden Sektoren. Der Zähler muß einen Wert ungleich 0 enthalten (1-255).

**Kontrollbyte**

Dieses Byte enthält verschiedene Angaben, je nach dem verwendeten Kommando.

Um nun einen so aufgebauten Kommando-Block an den HDC zu übertragen, muß folgendermaßen vorgegangen werden:

Zunächst wird der Prozessor durch die SUPER-Funktion ($20) des GEMDOS (TRAP #1) in den Supervisor-Modus geschaltet,

da einige privilegierte Zugriffe auf Hardware-Register erfolgen müssen.

Danach werden durch Setzen der System-Variablen FLOCK ($43E) die Routinen der Floppy-Bearbeitung gesperrt. Dies ist nötig, da sowohl der HDC als auch der FDC über die selben Hardware-Register gesteuert werden. Damit es dabei keine Überschneidungen geben kann, wie z.B. ein OK-Signal des einen Controllers, wenn auf das OK des anderen gewartet wird, wird der FDC quasi aus dem System entfernt und kann nicht mehr mitmischen.

In dem Hardware-Register $FF8606, im folgenden WDL genannt, werden durch Einschreiben des Wertes $88 die Bits 7 und 3 gesetzt, alle anderen gelöscht. Dadurch wird einerseits der HDC selektiert und andererseits die Leitung A1, die durch Bit 1 angesprochen wird, auf 0 gelegt.

Diese Leitung A1 dient zur Signalisierung an den HDC, daß nun ein Kommando-Byte (das erste Byte eines Kommando-Blocks) übertragen wird.

Danach wird im Register $FF8604, im folgenden WDC genannt, das Kommandobyte übergeben. Der HDC übernimmt dieses Byte und meldet dies durch ein 0-Signal an der HDC-Interrupt-Leitung. Diese Leitung liegt an Bit 5 des I/O-Portes des Multi-Funktions-Chip MFP und ist somit an der Adresse $FFFA01 zu finden.

Dieser Interrupt findet auch nach jedem weiteren übertragenen Byte statt. Kommt er nicht, so wurde das übertragene Byte entweder nicht erkannt oder der HDC ist nicht bereit, Daten zu empfangen.

Während der Übertragung der Kommando-Bytes wird auf den Interrupt maximal 100 Millisekunden lang gewartet, nach der volständigen Übertragung des Kommando-Blocks sogar bis zu 3 Sekunden, da dann ja das Kommando vollständig ausgeführt werden muß, bis der HDC ein OK melden kann. Ist nach dieser Zeit immer noch kein Interrupt erfolgt, so wird die Übertragung

der Kommando-Bytes abgebrochen und ein sogenanntes Timeout gemeldet.

Ist das Kommando-Byte übertragen und rechtzeitig durch den Interrupt quittiert, so wird eine $8A in das WDL geschrieben, wodurch die A1-Leitung wieder zu 1 wird.

Die restlichen 5 Bytes des Kommando-Blocks werden nun nach dem gleichen Schema, nur mit gesetzten Bit 1 (A1), übertragen. Zusammen mit einer $8A in WDL wird das Byte in WDC geschrieben, bis zu 100 Millisekunden auf den Interrupt gewartet (sonst Timeout) und dann das nächste Byte übertragen.

Nach Übertragung des letzten Bytes (Byte 5) des Kommando-Blocks wird nun länger (max. 3 Sekunden) auf den Interrupt gewartet, damit der HDC Zeit genug hat, das Kommando auszuführen.

Ist der Interrupt in der Zeit erfolgt, wird eine $80 in das WDL-Register geschrieben, um Bit 3 zurückzusetzen und damit den HDC zu deselektieren. Dadurch wird der FDC, der Floppy-Disk-Controller, wieder freigegeben.

Anschließend wird die System-Variable FLOCK ($43E) wieder auf 0 gesetzt, um die Floppy-Operationen wieder zu ermöglichen. Schließlich und endlich kann danach der Prozessor wieder in den User-Modus zurückgeschaltet werden.

Hier nun ein kleines Programm, welches die oben beschriebenen Schritte durchführt und einen Kommando-Block zum HDC sendet. Bitte beachten Sie, daß dies nur vollständig funktioniert, wenn das Kommando keine Datenübertragung über die DMA beinhaltet (z.B. READ, WRITE), da dann die DMA ebenfalls programmiert werden muß. Dazu kommen wir später.

```
; ** Hard-Disk-Zugriff S.D. **
; ** sendet Kommando-Bytes aus COM-Feld zum HDC **

wdc      = $ff8604
```

```
wdl          = $ff8606
wdcwdl       = wdc
port         = $fffa01
flock        = $43e
run:
             move.b     #'0',num       ;Timeout-Meldung vorbereiten
             clr.l      -(sp)
             move       #$20,-(sp)
             trap       #1             ;in Super-Modus umschalten
             addq.l     #6,sp
             move.l     d0,spsave      ;alten Stackpointer retten

             lea        com,a0         ;Zeiger auf Kommando-Block
             bsr        send           ;Kommando-Block an HDC senden
             bra        exit           ;fertig

send:                                 ;* Kommando-Block an HDC senden *
             st         flock          ;Floppy sperren
             move       #$88,wdl       ;HDC selektieren, A1=0
             clr.l      d0
             moveq      #5,d2          ;Zähler: 6 Bytes
loop:
             clr.l      d0
             move.b     (a0)+,d0       ;Byte holen
             bsr        send_byte      ;Byte an HDC senden
             bmi        error          ;Timeout !
             dbra       d2,loop        ;weitermachen

cont:
             move       #$8a,wdl
             bsr        waitl          ;max. 3s auf Interrupt warten
             bmi        error          ;Timeout !
             move       #$8a,wdl
             move       wdc,d0         ;Status-Byte holen
             move       #$80,wdl       ;HDC deselektieren
             move       wdc,d1         ;Completion-Byte holen
             clr        flock          ;Floppy freigeben
             rts                       ;fertig
```

```
exit:
        move.l      spsave,-(sp)
        move        #$20,-(sp)
        trap        #1              ;Umschalten in User-Modus
        addq.l      #6,sp
        rts                         ;Ende

error:                              ;Error melden
        clr         flock           ;Floppy freigeben
        move.l      #senderr,d0
        bsr         pline           ;Fehlermeldung ausgeben
        bra         exit            ;und Schluß

send_byte:                          ;* ein Byte zum HDC senden *
        swap        d0              ;Byte ins HI-Wort
        move        #$8a,d0         ;$8A ins LO-Wort
        move.l      d0,wdcwdl       ;WDC und WDL setzen
        bra         wait            ;warten auf OK (Interrupt)

waitl:
        add.b       #1,num          ;Durchlauf-Nummer+1
        move.l      #450000,d3      ;Timeout nach 3 Sekunden
        bra         wait1           ;warten...
wait:
        add.b       #1,num          ;Durchlauf-Nummer+1
        move.l      #15000,d3       ;Timeout nach 100 ms
wait1:
        subq.l      #1,d3           ;Timeout-Zähler-1
        bmi         timeout         ;Timeout !
        move.b      port,d0         ;I/O-Port laden
        and.b       #$20,d0         ;Bit 5 ausblenden
        bne         wait1           ;noch gesetzt, weiterwarten
        moveq       #0,d3           ;OK übergeben
        rts                         ;fertig
timeout:
        moveq       #-1,d3          ;nicht OK übergeben
        rts

pline:                              ;* Zeile auf Bildschirm ausgeben *
        move.l      d0,-(sp)
```

```
         move        #9,-(sp)
         trap        #1
         addq.l      #6,sp
         rts

spsave:  dc.l 0
senderr: dc.b "ERROR bei send_byte "
num:     dc.b "1. mal !",10,13,0
com:     dc.b $b,$0,$0,0,0,$0
even
```

Die Bytes des hier übertragenen Kommando-Blocks lassen den Schreib-/Lesekopf der Harddisk auf Spur 0 fahren ($B=Seek). Das Programm beinhaltet zu Testzwecken noch eine Fehler-Ausgabe, die ein Timeout mit Angabe des Zeitpunktes auf dem Bildschirm ausgibt. Dieser Teil kann natürlich entfallen, er dient nur zur Kontrolle der ordnungsgemäßen Übertragung des Kommando-Blocks.

Etwas komplizierter wird die Übertragung eines Schreib- bzw. Lese-Befehls an den HDC. Dabei muß zusätzlich zur Übertragung des Kommando-Blocks noch die DMA (Direct Memory Access) programmiert werden, die für die Übertragung der Daten zwischen Harddisk und Computer-Speicher zuständig ist. Die DMA benötigt folgende Informationen:

- die Speicher-Adresse, aus der bzw. in den die zu übertragenen Daten-Bytes zu lesen sind. Diese Adresse wird in den Hardware-Registern $FF8609, $FF860B und $FF860D übergeben, und zwar erst das LO-, dann das MID- und dann das HI-Byte der Adresse. Da dabei ein Byte zu einer vollständigen 32-Bit-Adresse fehlt, kann diese Adresse 'nur' in einem Bereich zwischen 0 und $FFFFFF liegen (s. auch FDC-Programmierung).

- die Richtung, in der die Daten zu übertragen sind, d.h. Lesen oder Schreiben. Diese Information erhält

die DMA aus Bit 8 des WDL-Wortes, eine 0 bedeutet Lesen in und eine 1 meint Schreiben aus dem Speicher.

- den Zustand, ob die DMA überhaupt eingeschaltet ist. Dies erfährt die DMA aus Bit 6 des WDL-Registers $FF8606. Normalerweise ist die DMA immer eingeschaltet, d.h. Bit 6=0.

Es ist auch wichtig, in welchem Moment der DMA diese Informationen gegeben werden, damit nicht Überschneidungen mit vorhergehenden DMA-Aktionen stattfinden können. Soll von der Harddisk gelesen werden, so wird erst das Kommando-Byte an den HDC übergeben und dann erst die DMA-Adresse gesetzt. Dadurch wird ausgeschlossen, daß die DMA unerwünschte Daten in den Speicher lädt, da der HDC nach Empfang des Kommandobytes erst auf die weiteren Bytes des Kommando-Blocks wartet.

Um auf die Festplatte zu schreiben, wird dagegen erst die DMA-Adresse gesetzt und danach das Kommando-Byte übertragen. Wie dies praktisch zu bewerkstelligen ist, können Sie anhand des Programmes 'HDC-Tools' im Kapitel 5.1.1.3 erkennen. Zunächst wollen wir jedoch bei der leidigen Theorie bleiben und die Kommandos für den HDC betrachten.

### 5.1.1.2 Liste der Befehle

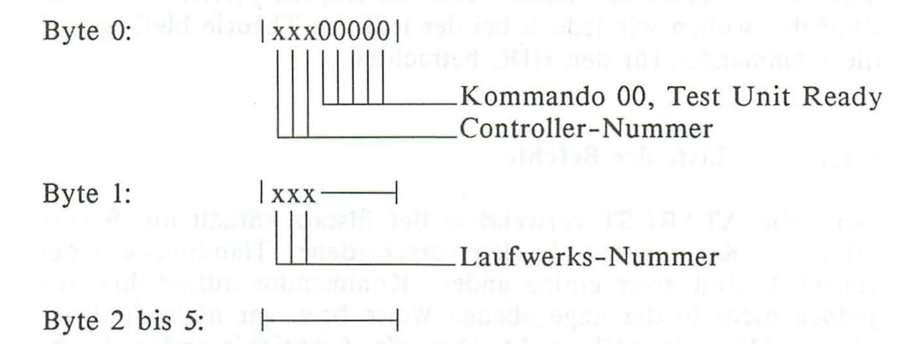
Der beim ATARI ST verwendete Befehlssatz enthält nur 9 verschiedene Kommandos. In den verschiedenen Handbüchern der Harddisk sind zwar einige andere Kommandos aufgeführt, die jedoch nicht in der angegebenen Weise bzw. gar nicht funktionieren. Hier eine Übersicht über die funktionierenden Kommandos mit dem dazugehörenden sedezimalen OpCode:

| OpCode | Kommando |
|---|---|
| 00 | Test Unit Ready |
| 01 | Restore |
| 03 | Request Sense |
| 04 | Format Drive |
| 08 | Read |
| 0A | Write |
| 0B | Seek |
| 15 | Mode Select |
| 1B | Seek to Shipping-Position |

Es folgt nun eine Erklärung der einzelnen Kommandos mit ihren Parameter-Bytes. Das '-'-Zeichen bedeutet, daß das Bit keine Bedeutung hat. Diese Bits sollten dann auf 0 gelegt werden.

**Test Unit Ready (00)**

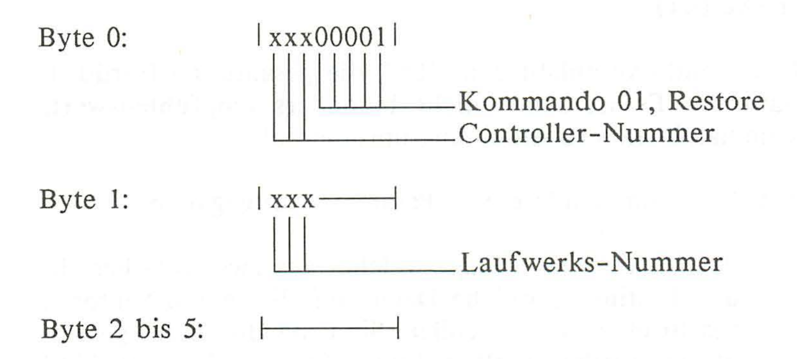
Mit diesem Kommando kann der Rechner den Bus ansprechen und feststellen, welche Geräte angeschlossen sind.




Ist das angegebene Laufwerk eingeschaltet und bereit, so wird im Status-Byte eine Null übergeben, andernfalls wird das Check Condition-Bit gesetzt.

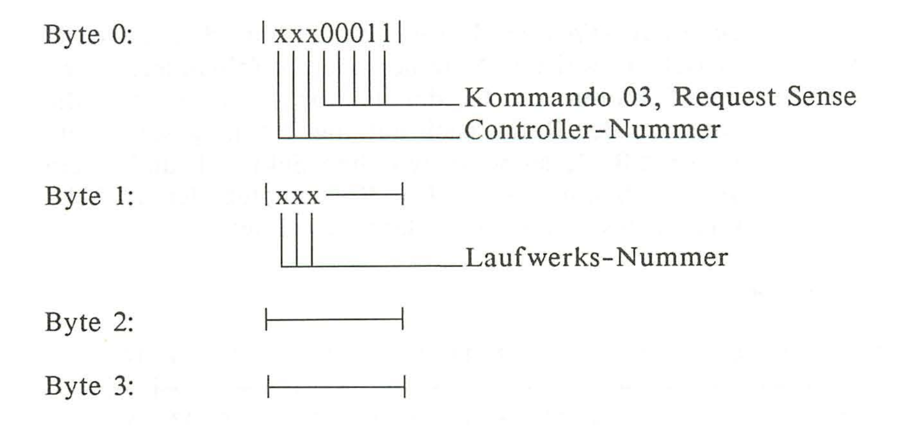
**Restore (01)**

Diese Kommando setzt den HDC in den Grundzustand zurück und läßt den Schreib-Lesekopf des Laufwerks zur Spur 0 zurückfahren.




**Request Sense (03)**

Dieses Kommando gibt 4 Bytes zurück (4 mal WDC auslesen!), wovon nur das erste Byte eine Bedeutung hat. Es enthält den Error-Code des zuletzt ausgeführten Kommandos. War kein Fehler aufgetreten, so erhält man dort eine 0.



Byte 4: |00000100| = $04 Bytes werden zurückgegeben

Byte 5: |————|

**Format Drive (04)**

Dieses Kommando veranlaßt den HDC, die gesamte (!) Harddisk zu formatieren. Es ist somit nicht besonders empfehlenswert, dieses Kommando experimentell auszuprobieren!

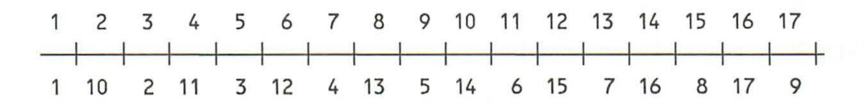
Es werden dem Kommando einige Parameter mitgegeben:

- das *Data-Pattern-Flag*, welches aus zwei Bits besteht und bestimmt, welche Daten auf die leeren Sektoren geschrieben werden sollen. Sind die Bits nicht gesetzt (0), so werden in alle Sektoren $6C geschrieben. Sind die Bits gesetzt, so wird das in Kommando-Byte 2 übergebene Byte geschrieben.

- *Data-Pattern:* Hier steht das Byte, mit dem bei gesetzten Data-Pattern-Flag die formatierten Sektoren gefüllt werden. Ist das Flag nicht gesetzt, so hat dieses Byte keine Bedeutung.

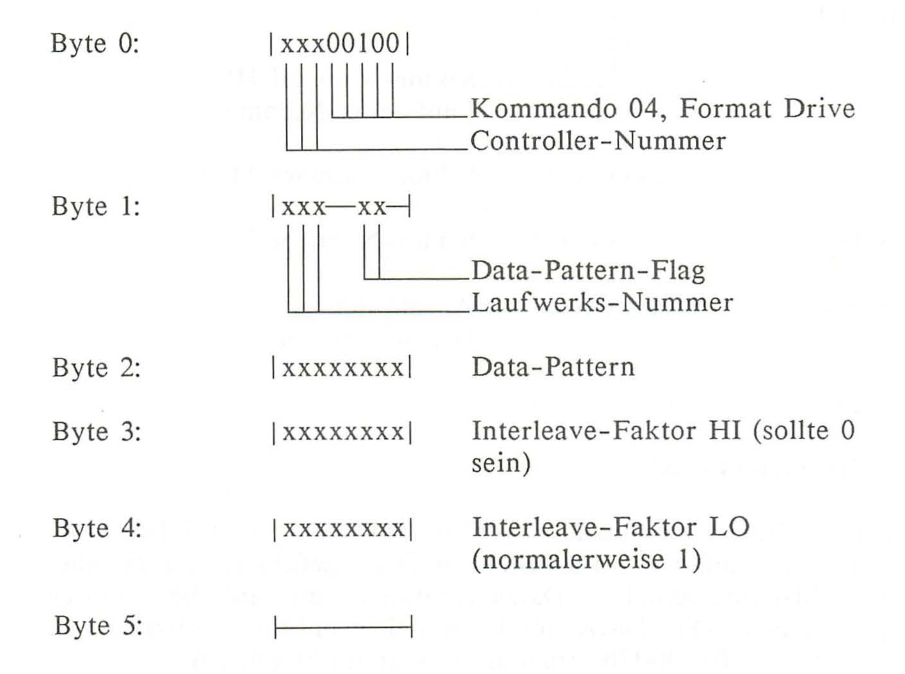
- *Interleave-Faktor:* Dieser Wert gibt den Abstand zwischen zwei der Nummer nach aufeinanderfolgenden Sektoren an. Ist der Faktor 1, so werden die Sektoren der Reihe nach auf die Tracks geschrieben. Ist er z.B. 2, so wird zwischen Sektor 1 und 2 ein anderer Sektor gelegt. Die Reihenfolge der 17 Sektoren eines Tracks wäre dann folgende:

laufende Nummer



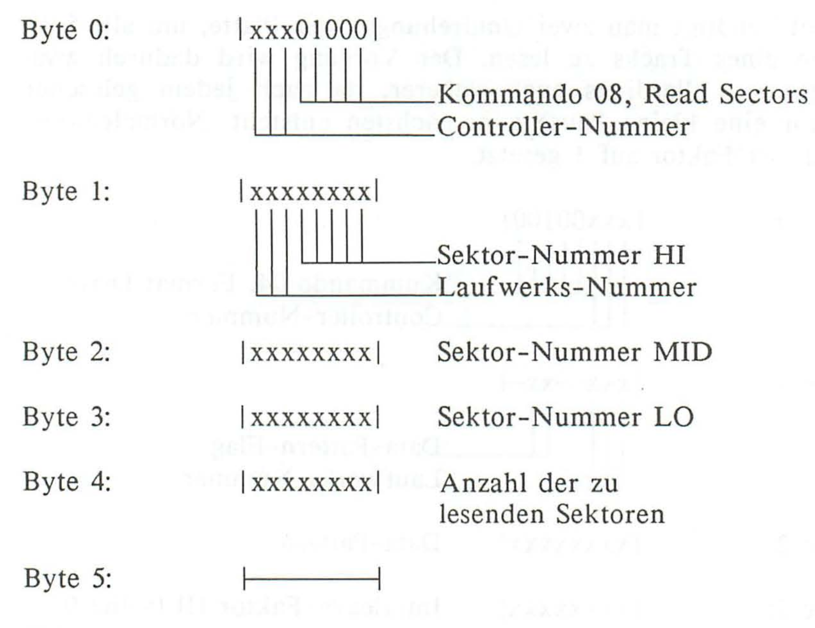

Sektor-Nummer

Somit benötigt man zwei Umdrehungen der Platte, um alle Sektoren eines Tracks zu lesen. Der Vorgang wird dadurch zwar langsamer, allerdings auch sicherer, da nach jedem gelesenen Sektor eine kleine Pause zum nächsten entsteht. Normalerweise ist dieser Faktor auf 1 gesetzt.



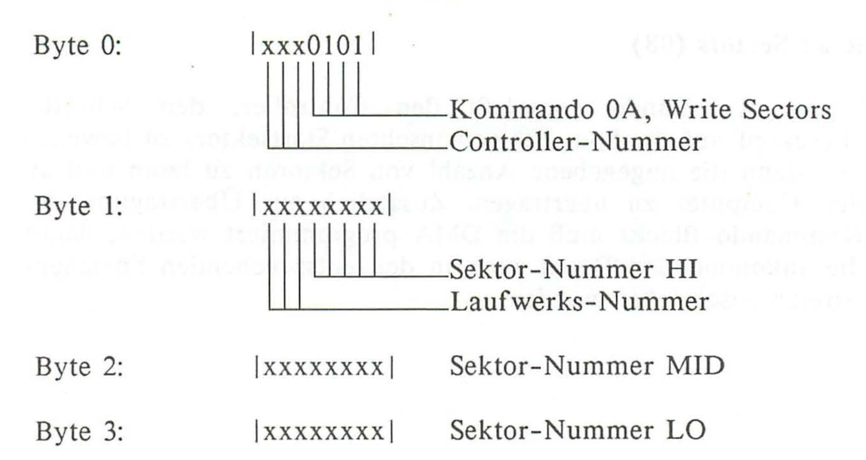

**Read Sectors (08)**

Dieses Kommando veranlaßt den Controller, den Schreib-/Lesekopf auf die Spur des gewünschten Startsektors zu bewegen und dann die angegebene Anzahl von Sektoren zu lesen und an den Computer zu übertragen. Zusätzlich zur Übertragung des Kommando-Blocks muß die DMA programmiert werden, damit die ankommenden Daten auch in den entsprechenden Speicherbereich geschrieben werden.




**Write Sectors (0A)**

Durch dieses Kommando werden Sektoren beschrieben. Der Kopf wird auf den entsprechenden Track gefahren und die über die DMA abgeschickten Daten empfangen und auf die Sektoren geschrieben. Die DMA muß hierbei ebenfalls zusätzlich zur Kommando-Block-Übertragung programmiert werden.

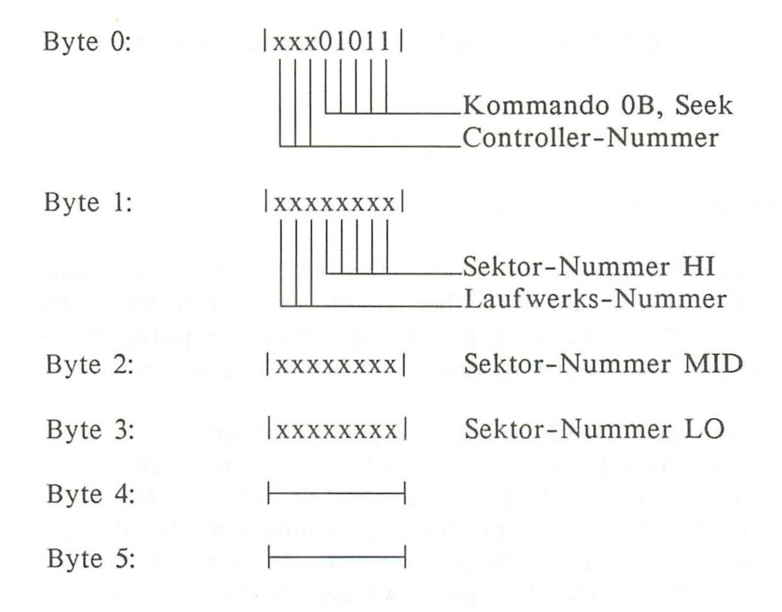


Byte 4: |xxxxxxxx| Anzahl der zu schreibenden Sektoren

Byte 5: |————|

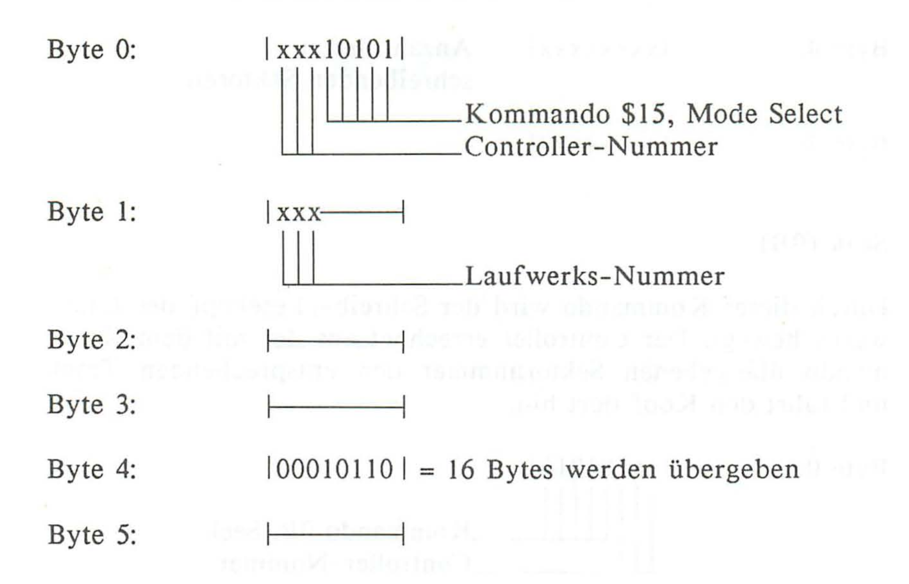
**Seek (0B)**

Durch dieses Kommando wird der Schreib-/Lesekopf des Laufwerks bewegt. Der Controller errechnet aus der mit dem Kommando übergebenen Sektornummer den entsprechenden Track und fährt den Kopf dort hin.




**Mode Select (15)**

Dieses Kommando wird zur Einstellung der Parameter für die Formatierung der Harddisk verwendet. Es wird nach der Übergabe des Kommandos (mit DMA-Programmierung !) ein 16-Byte-Block an den HDC übergeben.

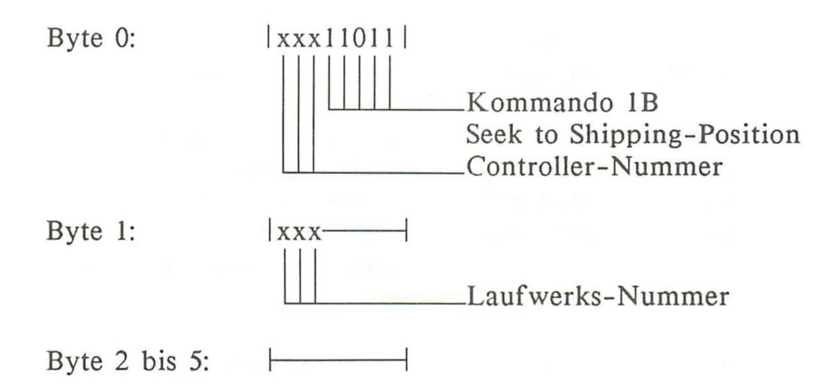



**Seek to Shipping-Position (1B)**

Durch dieses Kommando wird der Schreib-/Lesekopf auf eine Position gefahren, in der er sicher gegen Erschütterungen des Laufwerks ist. Diese Position nennt man auch 'Shipping-Position', da sie für den Transport des Laufwerks vorgesehen ist.

Das Programm SHIP.PRG läßt alle angeschlossenen Festplatten-Laufwerke auf diese Position fahren. Es sollte deshalb unbedingt vor dem Transport der Harddisk aufgerufen werden. Dabei ist zu beachten, daß beim Aufruf des Programms kein Inhaltsverzeichnis-Fenster auf dem Bildschirm ist, da sonst nach der Rückkehr aus dem SHIP-Programm wieder das Inhaltsverzeichnis der Festplatte ausgelesen wird. Dieser Zugriff auf die Festplatte wird aber den Kopf wieder aus der sicheren Position herausfahren.

Außerdem funktioniert das erste Kommando nach dem 1B-Kommando nicht korrekt, da der Kopf erst aus seiner Shipping-Position fahren muß.




### 5.1.1.3 HDC-Tools

Zur Demonstration des Schreib- bzw. Lesezugriffes auf die Harddisk nun ein Programm, welches wahlweise einen oder mehrere Sektoren liest und in den Speicher überträgt oder umgekehrt Sektoren mit Daten aus dem Speicher beschreibt. Im Beispiel werden 8 Sektoren ab Sektor 132 in den Speicher geladen. Dort liegt auch das Inhaltsverzeichnis des ersten Harddisk-Teils (Partition).

Außerdem ist in diesem Programm die einfache Übertragung eines Kommandoblocks integriert, ähnlich wie im Beispiel des Kapitels 5.1.1.1.

```
;** Harddisk-Sektor lesen/schreiben, Kommando senden **

wdc         = $ff8604 ;FDC/HDC-Access, DMA-Sector-Count
wdl         = wdc+2 ;DMA-Mode/Status
dma         = $ff8609 ;DMA-Adresse HI
flock       = $43e ;Floppy-VBL-Flag
port        = $fffa01 ;Parallel-Port, Bit 5=HDC-IRQ

run:
            clr.l -(sp)
            move        #$20,-(sp)
            trap        #1              ;in Supervisor-Mode schalten
```

```
        addq.l    #6,sp
        move.l    d0,spsave         ;User-Stackpointer retten
bp1:
        bra       put               ;für Nur-Übertragung
        pea       puffer            ;Puffer-Adresse
        move      #8,-(sp)          ;8 Sektoren
        move.l    #132,-(sp)        ;ab Sektor 132
        bsr       read              ;Sektor(en) in Puffer lesen
        bra       bp2
put:
        bsr       send              ;Kommando-Block übertragen
bp2:
        move.l    spsave,-(sp)
        move      #$20,-(sp)
        trap      #1                ;in User-Mode schalten
        addq.l    #6,sp

        rts                         ;Rückkehr zum Aufrufer
                                    ; oder
        clr       -(sp)
        trap      #1                ;Rückkehr zum Desktop

send:                               ;* Kommando-Block übertragen *
        lea       wdc,a0
        lea       com,a1            ;Zeiger auf Kommando-Block
        st        flock             ;Floppy sperren
        move      #$88,wdl          ;HDC selektieren, A1=0
        clr.l     d0
        moveq     #5,d1
loop:
        clr.l     d0
        move.b    (a1)+,d0
        bsr       send_byte         ;Byte an HDC senden
        bmi       tout              ;Timeout !
        dbra      d1,loop           ;sonst weitermachen

        bsr       waitl             ;max. 3 Sekunden warten
        bmi       tout              ;Timeout !
        move      wdc,d6
        move      #$80,wdl          ;sonst
```

```
        clr         flock             ;Floppy freigeben
        rts                           ;fertig

read:   ; * Sektor(en) lesen *
        lea         wdc,a0
        st          flock             ;Floppy-VBL-Routine sperren
        move        #$88,2(a0)        ;HDC-Zugriff, A1=0
        nop
        move.l      #$08008a,(a0)     ;READ-Kommando

        move.l      10(sp),-(sp)      ;Puffer-Adresse
        bsr         setdma            ;DMA setzen
        addq.l      #4,sp

        bsr         set_parameters    ;Sektoranzahl und -Nummer
        bmi         tout              ;Timeout aufgetreten !

        move        #$190,2(a0)
        nop
        move        #$90,2(a0)        ;Umschalten auf READ
        nop
        move        8(sp),(a0)        ;Sector-Count an DMA senden
        nop
        move        #$8a,2(a0)
        nop
        move.l      #0,(a0)           ;Übertragung starten
        bsr         waitl             ;max. 3 Sekunden warten
        bmi         tout              ;Timeout !
        move        #$8a,2(a0)
        bra         exec

write:  ; * Sektor(en) schreiben *
        lea         wdc,a0
        st          flock             ;Floppy-VBL sperren
        move.l      10(sp),-(sp)
        bsr         setdma            ;DMA-Adresse setzen
        addq.l      #4,sp

        move        #$88,2(a0)        ;HDC-Zugriff, A1=0
        nop
```

```
            move.l      #$0a008a,(a0)   ;WRITE-Kommando

            bsr         set_parameters  ;Sektoranzahl und -nummer
            bmi         tout            ;Timeout !

            move        #$90,2(a0)
            nop
            move        #$190,2(a0)     ;Umschalten auf WRITE
            nop
            move        8(sp),(a0)      ;Sector-Count an DMA senden
            nop
            move        #$18a,2(a0)
            nop
            move.l      #$100,(a0)      ;Übertragung starten
            bsr         waitl           ;max. 3 Sekunden warten
            bmi         tout            ;Timeout !
            move        #$18a,2(a0)

exec:
            nop
            move.l      (a0),d6         ;HDC/DMA-Status in D6 holen
            and.l       #$ff00ff,d6     ;HI=HDC, LO=DMA
tout:
            move        #$80,2(a0)      ;auf FDC umschalten
            nop
            move.l      (a0),d7         ;Completion-Byte holen
            and.l       #$ff00ff,d7     ;HI=HDC (0), LO=DMA
            clr         flock           ;Floppy-VBL-Routine freigeben
            rts                         ;fertig

set_parameters:     ;Sektor-Anzahl und Sector-Count setzen
            move        #$8a,2(a0)
            bsr         wait            ;warten auf HDC-OK
            bmi         setpx           ;Timeout !

            clr         d0
            move.b      4+5(sp),d0      ;Sektornr. HI
            bsr         send_byte
            bmi         setpx
```

```
            move.b      4+6(sp),d0      ;Sektornr. MID
            bsr         send_byte
            bmi         setpx

            move.b      4+7(sp),d0      ;Sektornr. LO
            bsr         send_byte
            bmi         setpx

            move        4+8(sp),d0      ;Anzahl der Sektoren
            bsr         send_byte
setpx:
            rts                         ;fertig

send_byte:  ; * 1 Byte zum HDC senden *
            swap        d0
            move        #$8a,d0
            move.l      d0,(a0)
            bra         wait

waitl:  ;max. 3 Sekunden auf OK warten
            move.l      #450000,count
            bra         wait1
wait:   ;max. 100 ms auf OK warten
            move.l      #15000,count
wait1:
            subq.l      #1,count
            bmi         timeout
            move.b      port,d0
            and.b       #$20,d0         ;HDC-Interrupt ?
            bne         wait1           ;nein
            moveq       #0,d0           ;ja => OK
            rts
timeout:
            move.l      #errline,d0
            bsr         pline           ;'Timeout' ausgeben
            moveq       #-1,d0          ;Timeout anzeigen
            rts

setdma:     ; * DMA-Adresse setzen *
            move.b      7(sp),dma+4     ;LO
```

```
            move.b      6(sp),dma+2         ;MID
            move.b      5(sp),dma           ;HI
            rts

pline:    ; * Zeile auf Bildschirm ausgeben *
            move.l      d0,-(sp)
            move        #9,-(sp)
            trap        #1
            addq.l      #6,sp
            rts

errline:    dc.b  "Timeout aufgetreten !",10,13,0
com:        dc.b  $b,0,0,132,0,0
even
count:      dc.l  1                         ;Timeout-Counter
spsave:     dc.l  0                         ;User-Stackpointer
puffer:     blk.b 512*8,$FF                 ;Puffer für 8 Sektoren
```

Mit diesem Programm ist man nun in der Lage, direkt Sektoren von der Harddisk zu laden bzw. zu schreiben. Die Status-Information, die eigentlich erfolgen sollte, wird in das Register D6 geschrieben, was bei Verwendung eines Debugger-Monitors (z.B. das SID-Programm oder K-SEKA) abgefragt werden kann.

Der Unterschied zu der über das Betriebssystem verfügbaren Sektoren-Lese-/Schreibe-Funktion liegt darin, daß dort nur auf den gewählten Teil der Harddisk, eben der Partition, zugegriffen werden kann. Will man aber nun z.B. den Sektor 0 der Harddisk lesen, so muß man dies mit obigem Programm tun.

### 5.1.1.4 Partitions-Analysator

Der erwähnte Sektor 0 ist auch recht interessant, da er ja Informationen über die Harddisk und ihre Partitionen enthält. Um diese Informationen zu lesen und auszuwerten, kann das nun folgende Programm verwendet werden. Es enthält u.a. auch Teile des obigen Programmes (read), so daß diese evtl. auch direkt übernommen werden können.

Das Programm liest auf die bekannte Weise den Sektor 0 der Harddisk aus und interpretiert die darin enthaltenen Daten. Diese werden dann auf dem Bildschirm ausgegeben, wobei alle Zahlen sedezimal (hexadezimal) dargestellt werden.

```
;** Partitions-Analysator S.D. **

wdc     = $ff8604                   ;FDC/HDC-Access, DMA-Sector-Count
wdl     = wdc+2                     ;DMA-Mode/Status
dma     = $ff8609                   ;DMA-Adresse HI
flock   = $43e                      ;Floppy-VBL-Flag
port    = $fffa01                   ;Parallel-Port, Bit 5=HDC-IRQ

run:
        lea       stp,sp
        clr.l     -(sp)
        move      #$20,-(sp)
        trap      #1                ;in Supervisor-Mode schalten
        addq.l    #6,sp
        move.l    d0,spsave         ;User-Stackpointer retten

        pea       puf               ;Puffer-Adresse
        move      #1,-(sp)          ;1 Sektor
        move.l    #0,-(sp)          ;ab Sektor 0
        bsr       read              ;Sektor(en) in Puffer lesen

        move.l    spsave,-(sp)
        move      #$20,-(sp)
        trap      #1                ;in User-Mode schalten
        addq.l    #6,sp

        move.l    #head,d0
        bsr       pline             ;Überschrift ausgeben

        move.l    #hi_cc,d0
        bsr       pmsg
        move      puf+$1b6,d0
        bsr       pword             ;Zylinder-Anzahl ausgeben
```

```
        bsr         pcrlf

        move.l      #hi_dhc,d0
        bsr         pmsg
        move.b      puf+$1b8,d0
        bsr         pbyt               ;Anzahl der Köpfe ausgeben
        bsr         pcrlf

        move.l      #hi_lz,d0
        bsr         pmsg
        move.b      puf+$1be,d0
        bsr         pbyt               ;Landeposition ausgeben
        bsr         pcrlf

        move.l      #hi_rt,d0
        bsr         pmsg
        move.b      puf+$1bf,d0
        bsr         pbyt               ;Seek-Rate ausgeben
        bsr         pcrlf

        move.l      #hi_in,d0
        bsr         pmsg
        move.b      puf+$1c0,d0
        bsr         pbyt               ;Interleave-Faktor ausgeben
        bsr         pcrlf

        move.l      #hi_spt,d0
        bsr         pmsg
        move.b      puf+$1c1,d0
        bsr         pbyt               ;Sektoren/Track ausgeben
        bsr         pcrlf

        move.l      #hd_size,d0
        bsr         pmsg
        move.l      puf+$1c2,d0
        bsr         plong              ;Sektoren auf Harddisk
                                       ;ausgeben
        bsr         pcrlf

        move.l      #bsl_count,d0
```

```
            bsr         pmsg
            move.l      puf+$1fa,d0
            bsr         plong           ;# tote Sektoren ausgeben
            bsr         pcrlf

            clr         d5
            clr.l       d6
            lea         puf+$1c6,a6     ;Partitions-Feld 0
loop:
            bsr         pcrlf
            move.b      d5,px_on
            add.b       #'0',px_on
            cmp.b       #0,0(a6,d6)     ;Partition aktiv ?
            bne         pon             ;ja
            move.l      #' aus',px_on+14  ;sonst 'aus' melden
            move.l      #px_on,d0
            bsr         pline
            bra         nextp
pon:
            move.l      #' an ',px_on+14
            move.l      #px_on,d0
            bsr         pline           ;'Partition an' ausgeben
            and.b       #$80,0(a6,d6)   ;Boot-bar ?
            beq         noboot          ;nein
            move.l      #boot,d0
            bsr         pline           ;sonst 'Boot-bar' ausgeben
noboot:
            move.b      1(a6,d6),px_id+18
            move        2(a6,d6),px_id+19
            move.l      #px_id,d0
            bsr         pline

            move.l      #px_start,d0
            bsr         pmsg
            move.l      4(a6,d6),d0
            bsr         plong           ;Startsektor ausgeben
            bsr         pcrlf

            move.l      #px_size,d0
            bsr         pmsg
```

```
            move.l      8(a6,d6),d0
            bsr         plong           ;Sektoren/Track ausgeben
            bsr         pcrlf

nextp:
            addq        #1,d5
            add         #12,d6
            cmp         #4*12,d6
            blt         loop

            move        #1,-(sp)
            trap        #1              ;warten auf Tastendruck
            addq        #2,sp
            clr         -(sp)
            trap        #1              ;Rückkehr ins Desktop

read:   ; Sektor(en) lesen (wie oben !)
            lea         wdc,a0
            st          flock           ;Floppy-VBL-Routine sperren
            move        #$88,2(a0)      ;HDC-Zugriff, A1=0
            nop
            move.l      #$8008a,(a0)    ;read-Command

            move.l      10(sp),-(sp)    ;Puffer-Adresse
            bsr         setdma          ;DMA setzen
            addq.l      #4,sp

            bsr         set_parameters  ;Sektoranzahl und -Nummer
                                         setzen
            bmi         tout            ;Timeout aufgetreten !

            move        #$190,2(a0)
            nop
            move        #$90,2(a0)      ;Umschalten auf READ
            nop
            move        8(sp),(a0)      ;Sector-Count an DMA übergeben
            nop
            move        #$8a,2(a0)
            nop
            move.l      #0,(a0)         ;Übertragung starten
```

```
                bsr         waitl
                bmi         tout
                move        #$8a,2(a0)
                move.l      (a0),d6             ;HDC/DMA-Status holen
                and.l       #$ff00ff,d6         ;HI=HDC, LO=DMA
tout:
                move        #$80,2(a0)          ;auf FDC umschalten
                nop
                move.l      (a0),d7             ;Completion-Byte holen
                and.l       #$ff00ff,d7         ;HI=FDC, LO=DMA
                clr         flock               ;Floppy-VBL-Routine freigeben
                rts                             ;fertig

set_parameters:   ;Sektor-Anzahl und Sector-Count setzen
                move        #$8a,2(a0)
                bsr         wait                ;warten auf HDC-OK
                bmi         setpx               ;Timeout !

                clr         d0
                move.b      4+5(sp),d0          ;Sektornr. HI
                bsr         send_byte
                bmi         setpx
                move.b      4+6(sp),d0          ;Sektornr. MID
                bsr         send_byte
                bmi         setpx
                move.b      4+7(sp),d0          ;Sektornr. LO
                bsr         send_byte
                bmi         setpx
                move        4+8(sp),d0          ;Anzahl der Sektoren
                bsr         send_byte
setpx:
                rts

send_byte:    ;1 Byte zum HDC senden
                swap        d0
                move        #$8a,d0
                move.l      d0,(a0)
                bra         wait
```

```
waitl:   ;max. 3 Sekunden auf OK warten
                move.l     #450000,count
                bra        wait1
wait:    ;max. 100 ms auf OK warten
                move.l     #15000,count
wait1:
                subq.l     #1,count
                bmi        timeout
                move.b     port,d0
                and.b      #$20,d0          ;HDC-Interrupt ?
                bne        wait1            ;nein
                moveq      #0,d0            ;ja => OK
                rts
timeout:
                move.l     #errline,d0
                bsr        pline
                moveq      #-1,d0           ;Timeout anzeigen
                rts

setdma:   ;DMA-Adresse setzen
                move.b     7(sp),dma+4      ;LO
                move.b     6(sp),dma+2      ;MID
                move.b     5(sp),dma        ;HI
                rts

; ** weitere Unterroutinen **

pline:   ; Print Line/CR
                bsr        pmsg
pcrlf:   ;Print CR,LF
                move       #10,d0
                bsr        pchar
                move       #13,d0
pchar:   ;Print Character D0
                move       d0,-(sp)
                move       #2,-(sp)
                trap       #1
                addq.l     #4,sp
                rts
```

```
pmsg:   ;Print Line (D0)
            move.l      d0,-(sp)
            move        #9,-(sp)
            trap        #1
            addq        #6,sp
            rts

plong:   ;D0 8-stellig hex ausgeben
            moveq       #7,d1
            bra         phexwl1
pword:   ;Print Hex-Word D0
            swap        d0
            moveq       #3,d1
            bra         phexwl1
pbyt:    ; Print Hex-Byte D0
            moveq       #1,d1
            ror.l       #8,d0
phexwl1:
            rol.l       #4,d0
            move.l      d0,-(sp)
            move.l      d1,-(sp)
            bsr         phexnib
            move.l      (sp)+,d1
            move.l      (sp)+,d0
            dbra        d1,phexwl1
            rts
phexnib:
            and.l       #$0f,d0
            add.b       #$30,d0
            cmp.b       #$3a,d0
            bcs         phexn1
            add.b       #7,d0
phexn1:
            bra         pchar             ;Zeichen ausgeben

head:     dc.b  "** Harddisk-Analyse 8/86 S.D. **",0
hi_cc:    dc.b  "Zylinder        : ",0
hi_dhc:   dc.b  "Köpfe           : ",0
hi_lz:    dc.b  "Park-Position   : ",0
hi_rt:    dc.b  "Seek-Rate       : ",0
```

```
hi_in:      dc.b "Interleave      : ",0
hi_spt:     dc.b "Sektoren/Track  : ",0
hd_size:    dc.b "Gesamt-Sektoren : ",0
bsl_count:  dc.b "tote Sektoren   : ",0
even
px_on:      dc.b "1. Partition :    ",0
boot:       dc.b "Boot-bar ",0
px_id:      dc.b "Partition-ID    :     ",0
px_start:   dc.b "Start-Sektor    : ",0
px_size:    dc.b "Sektoren-Anzahl : ",0

errline:    dc.b "Timeout aufgetreten !",10,13,0
even
data
count:      dc.l  1                     ;Timeout-Counter
spsave:     dc.l  0                     ;User-Stackpointer
            blk.l 200
stp:        blk.l 1
puf:        blk.b 512                   ;Puffer für einen Sektor
```

Und auch gleich das BASIC-Lader-Programm, welches das Programm 'ANAPART.TOS' auf der Diskette erstellt:

```
10   '*** Erstellung von ANAPART.TOS ***
15   '
20   ?:fullw 2:clearw 2:gotoxy 0,0
25   ? "File >> ANAPART.TOS << wird erzeugt":?:?:?
30   dim c%( 612):cs#=0
35   for i=0 to 612
40   read a$:c%(i)=val("&H"+a$)
45   check#=check#+(c%(i))
50   next i
55   if check#= 6435851 then 70
60   ?"Geht leider nicht, da etwas mit den DATA's nicht stimmt."
65   goto 80
70   bsave "anapart.tos",varptr(c%(0)), 1226
75   ? "Das Programm >> anapart.tos << ist nun geschrieben."
80   ?:?:?:?"Bitte Taste drücken":a=inp(2):end
```

```
85    '
90    '********* DATAs für anapart.tos *********
95    '
100   DATA 601A,0000,0484,0000,0000,0000,0000,0000
101   DATA 0000,0000,0000,0000,0000,0000,4FF9,0000
102   DATA 07AC,42A7,3F3C,0020,4E41,5C8F,23C0,0000
103   DATA 0488,4879,0000,07B0,3F3C,0001,2F3C,0000
104   DATA 0000,6100,01A0,2F39,0000,0488,3F3C,0020
105   DATA 4E41,5C8F,203C,0000,0354,6100,02AC,203C
106   DATA 0000,0377,6100,02BE,3039,0000,0966,6100
107   DATA 02C6,6100,0298,203C,0000,038A,6100,02A6
108   DATA 1039,0000,0968,6100,02B6,6100,0280,203C
109   DATA 0000,039D,6100,028E,1039,0000,096E,6100
110   DATA 029E,6100,0268,203C,0000,03B0,6100,0276
111   DATA 1039,0000,096F,6100,0286,6100,0250,203C
112   DATA 0000,03C3,6100,025E,1039,0000,0970,6100
113   DATA 026E,6100,0238,203C,0000,03D6,6100,0246
114   DATA 1039,0000,0971,6100,0256,6100,0220,203C
115   DATA 0000,03E9,6100,022E,2039,0000,0972,6100
116   DATA 0230,6100,0208,203C,0000,03FC,6100,0216
117   DATA 2039,0000,09AA,6100,0218,6100,01F0,4245
118   DATA 4286,4DF9,0000,0976,6100,01E2,13C5,0000
119   DATA 0410,0639,0030,0000,0410,0C36,0000,6800
120   DATA 6600,001A,23FC,2061,7573,0000,041E,203C
121   DATA 0000,0410,6100,01B2,6000,0070,23FC,2061
122   DATA 6E20,0000,041E,203C,0000,0410,6100,019A
123   DATA 0236,0080,6800,6700,000C,203C,0000,0423
124   DATA 6100,0186,13F6,6801,0000,043F,33F6,6802
125   DATA 0000,0440,203C,0000,042D,6100,016C,203C
126   DATA 0000,0445,6100,017E,2036,6804,6100,0182
127   DATA 6100,015A,203C,0000,0458,6100,0168,2036
128   DATA 6808,6100,016C,6100,0144,5245,0646,000C
129   DATA 0C46,0030,6D00,FF52,3F3C,0001,4E41,544F
130   DATA 4267,4E41,41F9,00FF,8604,50F9,0000,043E
131   DATA 317C,0088,0002,4E71,20BC,0008,008A,2F2F
132   DATA 000A,6100,00EA,588F,6100,0058,6B00,003C
133   DATA 317C,0190,0002,4E71,317C,0090,0002,4E71
134   DATA 30AF,0008,4E71,317C,008A,0002,4E71,20BC
135   DATA 0000,0000,6100,0076,6B00,0010,317C,008A
136   DATA 0002,2C10,0286,00FF,00FF,317C,0080,0002
```

```
137  DATA 4E71,2E10,0287,00FF,00FF,4279,0000,043E
138  DATA 4E75,317C,008A,0002,6100,0050,6B00,0030
139  DATA 4240,102F,0009,6100,0028,6B00,0022,102F
140  DATA 000A,6100,001C,6B00,0016,102F,000B,6100
141  DATA 0010,6B00,000A,302F,000C,6100,0004,4E75
142  DATA 4840,303C,008A,2080,6000,0010,23FC,0006
143  DATA DDD0,0000,0484,6000,000C,23FC,0000,3A98
144  DATA 0000,0484,53B9,0000,0484,6B00,0014,1039
145  DATA 00FF,FA01,0200,0020,6600,FFEA,7000,4E75
146  DATA 203C,0000,046B,6100,0020,70FF,4E75,13EF
147  DATA 0007,00FF,860D,13EF,0006,00FF,860B,13EF
148  DATA 0005,00FF,8609,4E75,6100,001A,303C,000A
149  DATA 6100,0006,303C,000D,3F00,3F3C,0002,4E41
150  DATA 588F,4E75,2F00,3F3C,0009,4E41,5C4F,4E75
151  DATA 7207,6000,000E,4840,7203,6000,0006,7201
152  DATA E098,E998,2F00,2F01,6100,000C,221F,201F
153  DATA 51C9,FFF0,4E75,0280,0000,000F,0600,0030
154  DATA 0C00,003A,6500,0006,0600,0007,6000,FFAA
155  DATA 2A2A,2048,6172,6464,6973,6B2D,416E,616C
156  DATA 7973,6520,2038,2F38,3620,2053,2E44,2E20
157  DATA 2A2A,005A,796C,696E,6465,7220,2020,2020
158  DATA 2020,203A,2000,4B94,7066,6520,2020,2020
159  DATA 2020,2020,2020,3A20,0050,6172,6B2D,506F
160  DATA 7369,7469,6F6E,2020,203A,2000,5365,656B
161  DATA 2D52,6174,6520,2020,2020,2020,3A20,0049
162  DATA 6E74,6572,6C65,6176,6520,2020,2020,203A
163  DATA 2000,5365,6B74,6F72,656E,2F54,7261,636B
164  DATA 2020,3A20,0047,6573,616D,742D,5365,6B74
165  DATA 6F72,656E,203A,2000,746F,7465,2053,656B
166  DATA 746F,7265,6E20,2020,3A20,0000,312E,2050
167  DATA 6172,7469,7469,6F6E,203A,2020,2020,0042
168  DATA 6F6F,742D,6261,7220,0050,6172,7469,7469
169  DATA 6F6E,2D49,4420,2020,203A,2020,2020,2020
170  DATA 0053,7461,7274,2D53,656B,746F,7220,2020
171  DATA 203A,2000,5365,6B74,6F72,656E,2D41,6E7A
172  DATA 6168,6C20,3A20,0054,696D,656F,7574,2061
173  DATA 7566,6765,7472,6574,656E,2021,0A0D,0000
174  DATA 0000,0002,1006,140E,0A0A,0E0A,0E0A,0E0A
175  DATA 0E0A,0E0A,0E0A,0E0A,120A,0814,0612,0614
176  DATA 0C08,060A,16FC,0E06,1C00
```

Wie bereits im Kapitel über Boot-Sektoren beschrieben, gibt das px_flag für jede Partition (max. 4) an, ob sie aktiv und bootbar ist. Die Harddisk des ATARI ST enthält üblicherweise keine boot-bare Partition, da der ST nicht von der Harddisk booten kann ohne den Harddisk-Treiber (AHDI.PRG).

Mit der Bezeichnung 'Seek-Rate' wird normalerweise eine 2 ausgegeben, was 3 Millisekunden pro Schritt (Step) bedeutet. Interleave kann von 1 bis 16 (Sektoren/Track-1) sein, beträgt hier jedoch üblicherweise 1. Dies stellt den Abstand zwischen zwei der Nummer nach aufeinanderfolgenden Sektoren auf dem Track dar, ebenso wie bei Disketten.

Der Wert hinter 'tote Sektoren' gibt die Anzahl der defekten Sektoren auf der gesamten Harddisk an. Diese Sektoren werden vom HDX.PRG-Programm erkannt und markiert. Eine 0 bedeutet hier, daß die Harddisk vollständig in Ordnung ist. Ansonsten ist ein defekter Sektor pro Megabyte noch ganz normal.

## 5.2 Anschluß der Festplatte

Die 19-polige Buchse auf der Rückseite des ST stellt die DMA-Schnittstelle dar. An diesem Anschluß wird die Festplatte mit dem beiliegenden (recht kurzen) Kabel angeschlossen und hat somit über den DMA-Chip eine direkte Verbindung mit dem Speicher des ST. Der Grund für die so kurze Strippe liegt übrigens in der immens hohen Datenübertragungsrate, die über dieses Kabel läuft. Längere Drähte wirken dabei wie Antennen, so daß sich dabei einige Signale von einem zum anderen Draht übertragen und somit den gesamten Datenaustausch stören können!
Die Datenübertragung läuft parallel über 8 Datenleitungen (Pins 1-8), so daß immer ein komplettes Byte auf einmal übertragen werden kann.

Zusätzlich verfügt diese Schnittstelle über verschiedene Service-Leitungen wie Reset (Pin 12), durch die der ST die Festplatte

bei RESET auf den Grundzustand zurücksetzt, oder eine Interrupt-Leitung (Pin 10), durch die die Harddisk sich dem ST bemerkbar machen und empfangene Daten quittieren kann.

Die gesamte Verbindung von Harddisk und ATARI ST läuft also nur über diesen Stecker ab. Theoretisch kann man hier sogar bis zu 8 Harddisk-Controller mit ihrerseits bis zu 8 Laufwerken anschließen, jedoch ist dies durch einen fehlenden Zweitanschluß an der Harddisk nicht ohne Bastelei möglich...

Um nun mit der Harddisk zu kommunizieren, muß der Rechner seine Wünsche in Form von Kommando-Blöcken über die Datenleitungen senden. Diese Kommando-Blöcke sind bereits beschrieben worden. Sie werden, wie auch die zu schreibenden bzw. zu lesenden Daten, über die 8 Bit der Datenleitungen übergeben. Dies wird im HDC-Tools-Programm durch einfache Anwahl des entsprechenden Registers und Einschreiben des Kommando-Bytes erreicht. Das Byte steht nun an den Datenleitungen an und kann von der Harddisk übernommen werden, die dies über die Interrupt-Leitung quittiert.

Um den Datenaustausch überhaupt zu ermöglichen, muß das Treiberprogramm AHDI.PRG (ATARI Hard-Disk-Interface) geladen werden. Dieses und das HDX-Programm laufen jedoch nur, wenn das TOS im Rechner eingebaut ist. Zwar läuft der Treiber auch sonst, aber nicht das HDX-Programm, ohne das die Arbeit mit der Festplatte unmöglich ist. Die Festplatte muß nämlich unbedingt vor der Benutzung formatiert und damit partitioniert werden, auch wenn Sie beim neuen Gerät schon eine Kapazität von 20 MByte vorfinden. Der Controller kann nämlich nur maximal 16 MByte pro Partition verarbeiten!

## 5.3 Komplettes Inhaltsverzeichnis ausdrucken

Um auch die große Anzahl der Dateien, die auf eine Harddisk passen, übersichtlich zu ordnen, verwendet man häufig Ordner, die auch entsprechend verschachtelt werden, d.h. Ordner enthalten wiederum Ordner. Das bringt zwar eine große Übersicht-

lichkeit, führt jedoch oft dazu, daß man nicht mehr genau weiß, welche Dateien in welchem Ordner oder 'Unter-Ordner' stecken.

Um dies herauszufinden, muß man immer wieder Ordner öffnen und schließen, um sich die Inhalte der vielen Ordner anzusehen. Viel praktischer wäre es, wenn man sich den gesamten Inhalt der Harddisk (oder auch einer Diskette) einfach ausdrucken könnte. Dies ist jedoch mit dem ATARI-Betriebssystem nicht ohne Klimmzüge möglich.

Es soll nun ein Programm vorgestellt werden, welches diese Aufgabe löst. Es fragt nach dem Einladen nach der Laufwerksbezeichnung (a-f) und gibt danach auf dem Drucker alle Dateien mit den entsprechenden Ordnern aus, die sich auf diesem Laufwerk (Diskette oder Harddisk) befinden. Dabei werden Ordner-Inhalte immer zwei Leerzeichen nach rechts eingerückt, so daß die Verschachtelung ebenfalls deutlich sichtbar wird.

Zusätzlich zum Namen wird jeweils die Länge der Dateien dezimal neben den Namen ausgegeben. Ein solcher Ausdruck kann zwar bei großer Belegung der Harddisk recht lang werden, lohnt sich jedoch sehr, um die Übersicht über seine Dateien zu behalten. Schließlich können Sie so auch erkennen, ob sie einige Dateien doppelt oder mehr auf der Platte liegen haben, was doch nur Platz frißt...

Hier nun das Programm, welches vollständig in Maschinensprache geschrieben ist, und zwar für den SEKA-Assembler. Bei Verwendung eines anderen Assemblers müssen ggf. die Kommentare statt mit einem Semikolon mit einem Sternchen (*) begonnen werden und die 'blk.x'-Anweisung durch 'ds.x' ersetzt werden.

```
;** Komplettes Inhaltsverzeichnis ausgeben 8/86 S.D. **

run:
        lea      stp,sp
        move.l   #menu,d0
        bsr      pmsg
        bsr      getkey          ;Laufwerk eingeben
        cmp      #'a',d0
        blt      run             ;falsches Laufwerk
        cmp      #'f',d0
        bgt      run             ;falsches Laufwerk

        move.b   d0,fname
        bsr      pcrlf
        lea      fname+7,a6      ;Zeiger auf Filenamen-Ende +1
        pea      dta
        move     #$1a,-(sp)
        trap     #1              ;SETDTA
        addq.l   #6,sp

        clr      d4              ;Tiefe 0
        lea      tiefen,a4       ;Zeiger auf Zähler()-Array
        move.b   #0,(a4)         ;Zähler=0
        bsr      sfirst
        bra      test

sfirst:
        move     #$10,-(sp)
        pea      fname
        move     #$4e,-(sp)
        trap     #1              ;SFIRST
        addq.l   #8,sp
sea:
        cmp.b    #'.',dta+30     ;Subdir ?
        bne      seax
        bsr      snext1
        tst      d0
        bne      seax
        bra      sea
```

```
seax:
        rts

snext:
        add.b    #1,(a4,d4)
snext1:
        move     #$4f,-(sp)
        trap     #1               ;SNEXT
        addq.l   #2,sp
        rts

next:
        bsr      snext
test:
        tst      d0
        bne      up               ;wieder eine Ebene hoch
        cmp.b    #$10,dta+21      ;Subdirectory ?
        bne      output           ;nein: Eintrag ausgeben
        bra      down

up:
        subq     #1,d4            ;Tiefe-1
        bmi      fertig           ;Fertig !
        sub      #6,a6
mlop:
        cmp.b    #'\',-(a6)
        bne      mlop
        bsr      addwc            ;"*.*",0 hinzufügen
        bsr      sfirst
        clr      d7
        move.b   (a4,d4),d7       ;Zähler(tiefe) in D7
        addq     #1,d7            ;Zähler+1
        move.b   #0,(a4,d4)
selop:
        subq     #1,d7
        beq      next             ;fertig mit dieser Ebene
        bsr      snext            ;Zähler(Tiefe)-ten Eintrag
                                  ;suchen
        bra      selop
```

```
down:
            move.l      #sub,a5
            bsr         druline
            move.l      #dta+30,a5
            bsr         druline
            bsr         drucr               ;CR drucken
            addq        #1,d4               ;Tiefe+1
            move.b      #0,(a4,d4)
            subq.l      #4,a6
            move        #13,d7
            lea         dta+30,a3
flop:
            move.b      (a3)+,d0
            beq         flopx
            move.b      d0,(a6)+            ;Filename als Pfad übertragen
            dbra        d7,flop
flopx:
            bsr         addwc               ;"\*.*",0 hinzufügen
bp:
            bsr         sfirst
            bra         test                ;nächste Tiefe absuchen

addwc:
            move.b      #'\',(a6)+
            move.b      #'*',(a6)+
            move.b      #'.',(a6)+
            move.b      #'*',(a6)+
            move.b      #0,(a6)+
            rts

output:     ;Eintrag ausgeben
            cmp.b       #8,$e1b             ;Alternate-Taste gedrückt ?
            bne         out1                ;nein
            bra         fertig              ;sonst Abbruch
out1:
            lea         dta+30,a0
            lea         outln,a5            ;Ausgabe-Zeile
            move        d4,d5
blop:
            move        #' ',(a5)+
```

```
                dbra        d5,blop
blop1:
                move.b      (a0)+,d0
                beq         blop1x
                move.b      d0,(a5)+
                bra         blop1
blop1x:
                move.b      #' ',(a5)+
                cmp.l       #outln+26,a5
                blt         blop1x
                move.l      dta+26,d0
                bsr         pdez8
                move.b      #0,(a5)
                move.l      #outln,a5
                bsr         druline
                bsr         drucr
                bra         next

fertig:   ;das war's
                clr         -(sp)
                trap        #1                  ;Exit => Desktop

menu:           dc.b  "** Inhaltsverzeichnis-Ausgabe S.D. **",10,13
                dc.b  "Bitte Laufwerk eingeben (a-f) :",0
sub:            dc.b  "Sub-Directory : ",0
fname:          dc.b  "a:\*.*",0,"                    ",0
even

; ** Unterprogramme **

getkey:   ;Get Key -> D0
                move        #1,-(sp)
                trap        #1
                and.l       #$ff,d0
                addq.l      #2,sp
                rts

pline:    ;Print Line/CR
                bsr         pmsg
```

```
pcrlf:  ;Print CR,LF
        move        #10,d0
        bsr         pchar
        move        #13,d0
pchar:  ;Print Character D0
        move        d0,-(sp)
        move        #2,-(sp)
        trap        #1
        addq.l      #4,sp
        rts

druline:  ;Zeile ab (a5) drucken
        move.b      (a5)+,d0
        beq         drux
        bsr         druchr              ;Zeichen drucken
        bra         druline
drux:
        rts

drucr:  ;CR/LF drucken
        move        #10,d0
        bsr         druchr
        move        #13,d0
druchr:
        move        d0,-(sp)
        move        #5,-(sp)
        trap        #1                  ;Zeichen drucken
        addq.l      #4,sp
        rts

pmsg:  ;Print Line (D0)
        move.l      d0,-(sp)
        move        #9,-(sp)
        trap        #1
        addq        #6,sp
        rts

pdez8:  ;D0 8-stellig dezimal ausgeben
        divu        #10000,d0
        swap        d0
```

```
          move      d0,-(sp)       ;Rest
          swap      d0
          and.l     #$ffff,d0
          move.l    #1000,d1
          bsr       dez1
          move      (sp)+,d0
pdez4:  ;D0 4-stellig dezimal ausgeben
          move.l    #1000,d1
dez1:
          divu      d1,d0
          move.l    d0,-(sp)
          add       #'0',d0
          move.b    d0,(a5)+       ;Zeichen in Ausgabezeile
          move.l    (sp)+,d0
          swap      d0
          and.l     #$ffff,d0
          divu      #10,d1
          bne       dez1
          rts

data
dta:      blk.b     44
temp:     blk.l      0
tiefen:   blk.b     10
outln:    blk.b     80
          blk.l    200
stp:      blk.l      1
```

Und auch hier noch der BASIC-Lader. Es wird das Programm 'ALLDIR.TOS' auf der Diskette erzeugt:

```
10   '*** Erstellung von ALLDIR.TOS ***
15   '
20   ?:fullw 2:clearw 2:gotoxy 0,0
25   ? "File >> ALLDIR.TOS << wird erzeugt":?:?:?
30   dim c%( 374):cs#=0
35   for i=0 to 374
40   read a$:c%(i)=val("&H"+a$)
```

```
45   check#=check#+(c%(i))
50   next i
55   if check#= 3796015 then 70
60   ?"Geht noch nicht, da etwas mit den DATAs nicht stimmt."
65   goto 80
70   bsave "ALLDIR.TOS",varptr(c%(0)), 749
75   ? "Das Programm >> ALLDIR.TOS << ist nun geschrieben."
80   ?:?:?:?"Bitte Taste drücken":a=inp(2):end
85   '
90   '********* DATAs für ALLDIR.TOS *********
95   '
100  DATA 601A,0000,02BC,0000,0000,0000,0000,0000
101  DATA 0000,0000,0000,0000,0000,0000,4FF9,0000
102  DATA 0666,203C,0000,01A2,6100,0266,6100,0212
103  DATA 0C40,0061,6D00,FFE6,0C40,0066,6E00,FFDE
104  DATA 13C0,0000,01F5,6100,020A,4DF9,0000,01FC
105  DATA 4879,0000,02BC,3F3C,001A,4E41,5C8F,4244
106  DATA 49F9,0000,02EC,18BC,0000,6100,0006,6000
107  DATA 0044,3F3C,0010,4879,0000,01F5,3F3C,004E
108  DATA 4E41,508F,0C39,002E,0000,02DA,6600,0010
109  DATA 6100,0014,4A40,6600,0006,6000,FFE8,4E75
110  DATA 0634,0001,4800,3F3C,004F,4E41,548F,4E75
111  DATA 6100,FFEE,4A40,6600,0012,0C39,0010,0000
112  DATA 02D1,6600,0096,6000,0038,5344,6B00,00EC
113  DATA 9CFC,0006,0C26,005C,6600,FFFA,6100,0066
114  DATA 6100,FF90,4247,1E34,4800,5247,19BC,0000
115  DATA 4800,5347,6700,FFBA,6100,FFA6,6000,FFF4
116  DATA 2A7C,0000,01E4,6100,0160,2A7C,0000,02DA
117  DATA 6100,0156,6100,0162,5244,19BC,0000,4800
118  DATA 598E,3E3C,000D,47F9,0000,02DA,101B,6700
119  DATA 0008,1CC0,51CF,FFF6,6100,000A,6100,FF34
120  DATA 6000,FF72,1CFC,005C,1CFC,002A,1CFC,002E
121  DATA 1CFC,002A,1CFC,0000,4E75,0C39,0008,0000
122  DATA 0E1B,6600,0006,6000,0052,41F9,0000,02DA
123  DATA 4BF9,0000,02F6,3A04,3AFC,2020,51CD,FFFA
124  DATA 1018,6700,0008,1AC0,6000,FFF6,1AFC,0020
125  DATA BBFC,0000,0310,6D00,FFF4,2039,0000,02D6
126  DATA 6100,00FA,1ABC,0000,2A7C,0000,02F6,6100
127  DATA 00B8,6100,00C4,6000,FEF8,4267,4E41,2A2A
128  DATA 2049,6E68,616C,7473,7665,727A,6569,6368
```

```
129  DATA 6E69,732D,4175,7367,6162,6520,2A2A,0A0D
130  DATA 4269,7474,6520,4C61,7566,7765,726B,2065
131  DATA 696E,6765,6265,6E20,2861,2D66,2920,3A00
132  DATA 5375,622D,4469,7265,6374,6F72,7920,3A20
133  DATA 0061,3A5C,2A2E,2A00,2020,2020,2020,2020
134  DATA 2020,2020,2020,2020,2020,2020,2020,2020
135  DATA 2020,2020,2020,2020,2020,2020,2020,2000
136  DATA 3F3C,0001,4E41,0280,0000,00FF,548F,4E75
137  DATA 613E,303C,000A,6104,303C,000D,3F00,3F3C
138  DATA 0002,4E41,588F,4E75,101D,6700,000A,6100
139  DATA 0014,6000,FFF4,4E75,303C,000A,6100,0006
140  DATA 303C,000D,3F00,3F3C,0005,4E41,588F,4E75
141  DATA 2F00,3F3C,0009,4E41,5C4F,4E75,80FC,2710
142  DATA 4840,3F00,4840,0280,0000,FFFF,223C,0000
143  DATA 03E8,6108,301F,223C,0000,03E8,80C1,2F00
144  DATA 0640,0030,1AC0,201F,4840,0280,0000,FFFF
145  DATA 82FC,000A,66E6,4E75,0000,0002,061E,0A06
146  DATA 1016,1036,440A,1C44,0620,0A0E,0000
```

# 6. Die RAM-Disk

Als Dritter im Bunde der Massenspeicher für den ATARI ST steht die RAM-Disk. Eine solche scheinbare Diskettenstation im Speicher stellt eine interessante und vor allem sehr schnelle Möglichkeit der Datenspeicherung dar. Wie funktioniert das?

Zunächst einmal brauchen wir einen Speicherbereich, der von keiner anderen Anwendung des Computers verwendet werden kann. Dort hinein legen wir die Daten, die sonst auf eine Diskette geschrieben würden. Der Vorteil liegt auf der Hand: Daten im Speicher hin- und herzuschieben ist für den 68000-Prozessor des ST eine leichte Sache und geht daher mit einer enormen Geschwindigkeit vor sich. Zusätzlich entfallen alle mechanischen Vorgänge, die eine Diskettenstation bremsen (Kopfpositionierung, Motor anlaufen lassen usw.). Das Ergebnis: eine RAM-Disk ist sehr schnell.

Was wir noch brauchen, ist ein Programm. Dieses Programm muß die Verwaltung des RAM-Disk-Speichers übernehmen und die Daten je nach Bedarf im Speicher verschieben. Solche Programme sind bereits mehrere auf dem Markt, einige sind auch in der Literatur zu finden (z.B. Tips & Tricks ATARI ST). Sie arbeiten alle nach dem gleichen Prinzip, welches nun betrachtet werden soll.

Zuerst muß der als RAM-Disk zu verwendende Speicher initialisiert werden. Dazu muß ein Boot-Sektor erstellt werden, der alle Informationen über die Art, Aufteilung und Größe der 'Disk' enthält. Dieser Sektor ist auf richtigen Disketten der allererste, also müssen diese Parameter an den Anfang des RAM-Disk-Speichers geschrieben werden.

Danach muß sich das Programm installieren, d. h. es muß Vorbereitungen treffen, damit es immer erfährt, ob eine Datenübertragung stattfinden soll und wenn ja, wohin und in welcher Richtung. Dies wird erreicht, indem drei Zeiger des Betriebssystems auf eigene Routinen gerichtet werden. Diese Zeiger sind Speicherzellen, in denen Adressen von Programmen stehen. Will

das Betriebssystem ein solches Programm aufrufen, so wird der entsprechende Zeiger ausgelesen und zur erhaltenen Programmroutine verzweigt.

Die Zeiger, die bei der Installierung einer RAM-Disk verwendet werden, sind für die Bedienung der Hard-Disk vorgesehen. Sie liegen an den Speicheradressen $472 bis $47E und zeigen auf Routinen folgender Bedeutungen:

| Adresse | Name | Bedeutung |
|---|---|---|
| $472 | hdv_bpb | Feststellung und Übergabe des Parameterblockes, der Angaben über die Diskette bzw. Harddisk enthält. |
| $476 | hdv_rw | Schreib-/Leseroutine für die Harddisk. Dort wird der Datentransfer abgewickelt. |
| $47A | hdv_boot | Boot-Routine für die Harddisk. Wird von der RAM-Disk nicht benötigt, da von ihr nicht gebootet werden kann. |
| $47E | hdv_mediach | Feststellung, ob das Medium (Diskette) zwischenzeitlich gewechselt wurde. |

Sind die Zeiger schließlich alle neu eingestellt und ihre alten Inhalte gerettet, so kann das Programm verlassen werden. Dafür wird jedoch ein spezieller Aufruf des BIOS verwendet, mit dem sich ein bestimmter Speicherplatz reservieren läßt. Damit ist die RAM-Disk schließlich installiert.

Nun muß man noch ein Diskettensymbol des Desktop für die RAM-Disk vorbereiten. Dazu klickt man irgendeines der Diskettensymbole an und ändert nach Wahl des Menüpunktes 'Disk anmelden' den Namen und den Kennbuchstaben der Disketten-

station. Nach Wahl des 'OK'-Knopfes erscheint nun ein weiteres Symbol auf dem Bildschirm. Dieses kann nun nach gewohnter Manier zum Laden und Speichern von Daten und Programmen verwendet werden. Lediglich die Funktionen Formatieren und Disk-Copy funktionieren nicht, so daß nur einzelne Files bearbeitet oder gelöscht werden können.

Will nun das Betriebssystem auf die Hard- oder RAM-Disk zugreifen, so wird über einen der oben erwähnten Zeiger in das immer noch im Speicher stehende RAM-Disk-Programm gesprungen. Dort wird dann geprüft, ob sich die RAM-Disk angesprochen fühlen soll oder nicht. Wenn nicht, so wird zu der eigentlichen Routine verzweigt, deren Adresse ja gerettet wurde.

Ist doch die RAM-Disk gemeint, so beginnt das Programm mit seiner Arbeit. Bei einem Schreib-/Lesezugriff werden die Parameter wie Sektor, Anzahl der zu lesenden Sektoren und Datenübertragungs-Richtung vom Stack gelesen und die entsprechenden Daten im Speicher kopiert.

Handelt es sich um die 'Media-Change'-Anfrage, bei der nach einem eventuellen Wechsel des Speichermediums gefragt wird, so gib das RAM-Disk-Programm grundsätzlich eine 0 zurück. Diese bedeutet, daß nichts gewechselt wurde, was ja bei einer RAM-Disk auch nicht möglich ist.

Die dritte Art des Aufrufes bedeutet, daß das Betriebssystem die Speicheradresse des Parameterblockes erfahren möchte. Dafür wird die gewünschte Adresse im Register D0 übergeben.

Das waren schon alle Aufgaben eines RAM-Disk-Programmes. Was es jedoch nicht kann, ist die Erhaltung der Daten nach dem Abschalten des Rechners. Das ist nämlich der Haken bei der Sache: wirklich abgespeichert sind die Daten nicht. Aus diesem Grunde müssen die Daten, die z.B. mit einem Texteditor erstellt wurden, unbedingt vor dem sicher verdienten Feierabend noch von der RAM-Disk auf eine wirkliche Diskette bzw. Harddisk kopiert werden!

Doch nun genug der bloßen Theorie. Nun wollen wir uns ein RAM-Disk-Programm ansehen, in dem alle diese Sachen vorkommen.

## 6.1 Ein komfortables RAM-Disk-Programm

Das in diesem Kapitel vorgestellte Programm enthält einige Merkmale, die für die reine Anwendung einer RAM-Disk eigentlich nicht nötig wären. Da sie jedoch recht nützlich sind, ist das Programm zwar etwas umfangreicher, dafür aber komfortabler. Es ist für die Verwendung der RAM-Disk als Laufwerk C vorgesehen, läßt sich jedoch auch leicht auf eine andere Laufwerksnummer anpassen.

Das Programm ist ein Accessory, welches nach dem Booten in dem 'Desk'-Menü unter dem Punkt 'RAM-Disk' auftaucht. Wird dieser Menüpunkt angeklickt, so öffnet sich ein kleines Dialogfenster, welches drei Auswahlpunkte beinhaltet.

Der erste Punkt, welcher auch stark umrandet ist, trägt die Bezeichnung 'Exit'. Wenn man nun diesen Punkt anklickt oder auch nur die Return-Taste betätigt, so wird das Fenster gelöscht und das Desktop meldet sich wieder. Passiert ist dabei gar nichts. Dieser Punkt ist nur dafür vorgesehen, wenn der Menüeintrag 'RAM-Disk' versehentlich angewählt wurde. Dann ist 'Exit' der Notausgang.

Im mittleren Auswahlpunkt steht die Inschrift 'mehr'. Ein Anklicken dieses Knopfes verändert die Zahl in dem rechten Auswahlkästchen. Diese Zahl bedeutet die Größe der zu installierenden RAM-Disk. Durch Anwahl von 'mehr' wird diese Zahl in 100er-Schritten erhöht, bis nach dem Wert 800 wieder eine Null angeboten wird.

Hat man die gewünschte RAM-Disk-Kapazität eingestellt, so wählt man den Knopf mit der Zahl an. Da für die Installierung eines neuen Speicherbereiches der alte Inhalt der RAM-Disk gelöscht wird, erscheint zur Sicherheit ein weiteres Dialog-Fenster. In diesem Fenster muß auf die Frage 'Alten Inhalt der

RAM-Disk löschen?' mit Ja geantwortet werden; andernfalls bleibt die alte RAM-Disk mit ihrer alten Kapazität und Belegung erhalten.

Nachdem durch die Wahl von 'Ja' nun alle Einstellungen erledigt sind, macht sich das Programm an die Arbeit. Als erstes wird der Speicherbereich, den die RAM-Disk vorher belegt hatte, wieder an das Betriebssystem zurückgegeben.

Danach versucht das Programm, den gewünschten Speicherbereich für sich zu reservieren. Sollte nicht genügend Speicherplatz vorhanden sein, so erhält man die Meldung 'Nicht genug RAM'. Nachdem diese Meldung quittiert wurde, ist sowohl die Meldung als auch die RAM-Disk verschwunden. Man muß dann einen kleineren Speicher auswählen, indem man nach erneuter Wahl des 'RAM-Disk'-Menüpunktes so oft die 'mehr'-Taste betätigt, bis der neue gewählte Wert erscheint.

Wählt man als Kapazität der RAM-Disk die Null an, so wird sie vollständig abgemeldet und belegt somit keinen Speicher mehr. Somit ist man mit diesem Programm in der Lage, seine RAM-Disk je nach Bedarf beliebig oft in der Größe verändern oder ab- und anmelden. Diese Möglichkeit haben die meisten auf dem Markt befindlichen RAM-Disk-Programme nicht, und bei häufigem Arbeiten mit diesem Programm werden Sie die Vorteile zu schätzen wissen.

Noch ein Punkt sollte erwähnt werden, bevor wir uns das Programm selbst ansehen. Da eine RAM-Disk nicht formatiert werden kann (bitte erst gar nicht probieren, da dies die Diskettenstationen ansprechen kann...), muß zum Löschen einer solchen 'Disk' jedes File einzeln gelöscht werden. Bei dem vorliegenden Programm jedoch brauchen Sie lediglich in dem Dialogfenster dieselbe Kapazität auszuwählen, und schon ist die gesamte RAM-Disk leer.

Doch nun zum Programm:

```
;***** RAM-Disk mit Komfort S.D. *****

hdv_bpb       = $472
hdv_rw        = $476
hdv_mediach   = $47e

drvbits       = $4c2

start:
              move.l      #nstapel,a7       ;neuen Stack einstellen

              move        #10,opcode        ;appl_init
              move        #0,sintin
              move        #1,sintout
              move        #0,saddrin
              move        #0,saddrout
              bsr         aes
              move        intout,appid      ;Application-ID

              move        #77,opcode        ;graf_handle
              move        #5,sintout
              move        #0,saddrin
              move        #0,saddrout
              bsr         aes
              move        intout,grhandle   ;Graphic-Handle

              move        #35,opcode        ;Menu_Register
              move        #1,sintin
              move        #1,sintout
              move        #1,saddrin
              move        appid,intin
              move.l      #accname,addrin
              bsr         aes
              move        intout,accid      ;Accessory-Nummer

 ;** Ab hier die Bereitschafts-Schleife **

loop:         bsr         event             ;Event_Multi
              cmp         #40,msgbuff       ;Acc_open ?
              bne         loop              ;nein
```

```
        move        msgbuff+8,d0
        cmp         accid,d0          ;unsere Accessory-Nummer ?
        bne         loop              ;nein
        bsr         run               ;Menü darstellen
        bra         loop              ;immer wieder...

        ; ** Auswahl **

run:
        move.l      #wieviel,addrin
        bsr         formalert         ;Auswahl darstellen
        move        intout,choice
        cmp         #1,choice         ;Exit?
        beq         ende              ;ja => Ende
        cmp         #3,choice         ;OK ?
        beq         ok                ;ja
        addq        #2,size           ;andere Größe anbieten
        cmp         #18,size          ;über 800 KByte?
        blt         more              ;nein
        clr         size              ;ja, wieder 0 KByte
more:
        lea         sizes,a0
        clr.l       d0
        move        size,d0
        move        0(a0,d0),kapazi   ;neue Größe einstellen
        lsl         #1,d0
        lea         dezi,a0
        move.l      0(a0,d0),offer    ;neue Größe anzeigen
        bra         run               ;wiederholen

 ; * Speicher reservieren *

ok:
        move.l      #clear,addrin
        bsr         formalert         ;wirklich löschen?
        cmp         #2,intout
        beq         okx               ;nein => Ende
        bsr         mfree             ;Speicher freigeben
        tst         size              ;0 KByte ?
        bne         ok1               ;nein
```

```
okx:
         rts                          ;0 Kbyte: fertig
ok1:
         move     #2,changed          ;'Diskette wird gewechselt'
         clr.l    d7
         move     kapazi,d7           ;Kapazität in KByte
         add.l    #9,d7               ;plus 9K für Verwaltung
         asl.l    #5,d7
         asl.l    #5,d7               ;mal 1024=Kapazität in Byte
         move.l   d7,-(sp)            ;anzumeldender RAM-Bereich
         move     #$48,-(sp)          ;MALLOC-Funktion
         trap     #1
         addq.l   #6,sp
         tst.l    d0                  ;Fehler aufgetreten ?
         beq      fehler              ;ja => Fehlermeldung
         move.l   d0,puffer           ;Startadresse der RAM-Disk
                                      ;retten

         move.l   #init,-(sp)
         move     #38,-(sp)           ;Initialisierung im
                                      ;Supervisor
         trap     #14
         addq.l   #6,sp
         rts
fehler:
         move.l   #error,addrin
         bsr      formalert           ;'Nicht genug RAM !'
         bra      ende                ;Abbruch

init:
         move.l   hdv_bpb,bpbsave     ;alte Vektoren retten
         move.l   #bpb,hdv_bpb
         move.l   hdv_rw,rwsave       ;Vektoren auf neue Routinen
                                      ;setzen
         move.l   #rw,hdv_rw
         move.l   hdv_mediach,mediasave
         move.l   #media,hdv_mediach

         move.l   puffer,a0
```

```
            move.l      #10240/4,d0
iloop1:
            clr.l       (a0)+           ;Boot-Sektor + FATs löschen
            dbra        d0,iloop1

 ; * Boot-Sektor generieren *
            move.l      puffer,a0
            add.l       #11,a0          ;ab Puffer+11
            lea         boottab,a1
            moveq       #tabend-boottab-1,d0
bloop:
            move.b      (a1)+,(a0)+    ;Daten in Boot-Sektor kopieren
            dbra        d0,bloop

            move        kapazi,d7
            move        d7,numcl        ;Kapazität in KByte in BPB

            lsl         #1,d7           ;Kapazität in Sektoren
            add         #18,d7          ;plus 18 Sektoren
            move.l      puffer,a0
            add.l       #19,a0          ;in Puffer+19 und +20
            move.b      d7,(a0)+        ;LO
            lsr         #8,d7
            move.b      d7,(a0)         ;HI

            bset        #2,drvbits+3    ;Drive C anmelden
            rts                         ;fertig

 ;* Funktion: Get BPB *

bpb:        cmp         #2,4(sp)        ;Drive C ?
            beq         bpb1            ;ja
            move.l      bpbsave,a0      ;alte Routine
                        jmp(a0)         ;aufrufen

bpb1:       move.l      #bpbtab,d0      ;Zeiger auf BIOS-Parame-
                                        ;ter-Block
            rts
```

```
;* Funktion: Read/Write *

rw:        cmp        #2,14(sp)        ;Drive C ?
           beq        rw1              ;ja
           move.l     rwsave,a0        ;alte Routine
           jmp        (a0)             ;aufrufen

rw1:       move       12(sp),d0        ;recno, logische Sektor Nummer
           ext.l      d0
           lsl.l      #8,d0
           lsl.l      #1,d0            ;mal 512

           move.l     6(sp),a0         ;Puffer-Adresse
           move       10(sp),d1        ;Anzahl Sektoren
           subq       #1,d1
           move.l     puffer,a1        ;Basis-Adresse
           add.l      d0,a1            ;plus relative Adresse in
                                       ;RAM-Disk

           move       4(sp),d0         ;rwflag
           btst       #0,d0            ;lesen ?
           beq        rloop0           ;ja
           exg        a0,a1            ;Ziel und Quelle tauschen

rloop0:    move.l     #511,d0          ;ein Sektor
rloop:     move.b     (a1)+,(a0)+      ;Puffer kopieren
           dbra       d0,rloop
           dbra       d1,rloop0        ;nächster Sektor
           clr        d0               ;OK
           rts

;* Funktion: Media-Change *

media:     cmp        #2,4(sp)         ;Drive C ?
           beq        media1           ;ja
           move.l     mediasave,a0     ;alte Routine
           jmp        (a0)             ;aufrufen

media1:    move       changed,d0       ;Diskette evtl. gewechselt
           clr        changed          ;aber nur einmal
```

```
            rts

event:
            move        #25,opcode        ;Event_Multi, GEM-Ereignis
                                          ;feststellen
            move        #16,sintin
            move        #7,sintout
            move        #1,saddrin
            move.l      #msgbuff,addrin
            lea         table,a1
            lea         intin,a2
            moveq       #15,d0
lop1:
            move        (a1)+,(a2)+       ;Parameter setzen
            dbra        d0,lop1
            bsr         aes
            rts

aes:    ; AES-Aufruf
            move.l      #aespb,d1
            move        #$c8,d0
            trap        #2
            rts

mfree:    ; Speicher freigeben
            tst.l       puffer
            beq         ende              ;ist bereits abgemeldet

            move.l      #reinit,-(sp)
            move        #38,-(sp)         ;Reinitialisierung
            trap        #14               ;im Supervisor-Modus
            addq.l      #6,sp

            move.l      puffer,-(sp)
            move        #$49,-(sp)        ;MFREE-Funktion, Speicher
                                          ;freigeben
            trap        #1
            addq.l      #6,sp
            tst.l       d0                ;Error?
            beq         ende              ;nein
```

```
            move.l      #error1,addrin
            bsr         formalert           ;Error-Meldung
ende:
            clr.l       puffer      ;kein Speicher mehr reserviert
            rts

reinit:
            move.l      bpbsave,hdv_bpb
            move.l      rwsave,hdv_rw           ;Vektoren auf alte
                                                ;Routinen setzen
            move.l      mediasave,hdv_mediach
            bclr        #2,drvbits+3            ;Drive C abmelden
            rts

formalert:
            move        #52,contrl          ;form_alert, Alarmfenster
                                            ;darstellen
            move        #1,contrl+2
            move        #1,contrl+4
            move        #1,contrl+6
            move        #0,contrl+8
            move        #1,intin
            bsr         aes
            rts

table:      dc.w  $13,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0
accname:    dc.b  " RAM-Disk ",0
even
wieviel:    dc.b  "[1][Wieviel RAM-Disk-KByte ?]"
            dc.b  "[Exit| mehr |"
offer:      dc.b  " 100 ]",0,0
clear:      dc.b  "[1][Alten Inhalt der| RAM-Disk löschen ?]"
            dc.b  "[ Ja ! | Nein ]",0,0
error:      dc.b  "[2][Nicht genug RAM !]"
            dc.b  "[Dann nicht..]",0,0
error1:     dc.b  "[2][Fehler bei MFREE !]"
            dc.b  "[Dann nicht..]",0,0
even
kapazi:     dc.w  100
size:       dc.w  2
```

```
sizes:      dc.w  0,100,200,300,400,500,600,700,800
dezi:       dc.b  '  0  100 200 300 400 500 600 700 800'

puffer:     dc.l  0                ;RAM-Disk-Puffer-Adresse
changed:    dc.w  0                ;Flag für 'Diskette gewechselt'

bpbtab:
recsiz:     dc.w  $200             ;Sektorgröße
clsiz:      dc.w  2                ;Clustergröße in Sektoren
clsizb:     dc.w  $400             ;Clustergröße in Bytes
rdlen:      dc.w  7                ;Directorylänge in Sektoren
fsiz:       dc.w  5                ;FAT-Größe
fatrec:     dc.w  6                ;FAT-Sektoren
datrec:     dc.w  18               ;Sektoren für Verwaltung
numcl:      blk.w 1                ;Kapazität in KByte
flags:      dc.l  0,0,0,0

boottab:   ; Daten in 8086-format
            dc.b  0,2              ;Bytes pro Sektor
            dc.b  2                ;Sektors pro Cluster
            dc.b  1,0              ;reserved Sektors
            dc.b  2                ;FATs
            dc.b  112,0            ;Directory entries
            blk.b 2                ;Sectors on media
            dc.b  0                ;media descriptor
            dc.b  5,0              ;Sectors pro FAT
            dc.b  9,0              ;Sectors pro Track
            dc.b  1,0              ;Sides
            dc.b  0                ;hidden
tabend:

even

bpbsave:    blk.l 1                ;Platz für alte Vektoren
rwsave:     blk.l 1
mediasave:  blk.l 1

aespb:      dc.l contrl,global,intin,intout,addrin,addrout

data
```

```
choice:     blk.w 1          ;Auswahl
grhandle:   blk.w 1
appid:      blk.w 1          ;Application-ID
accid:      blk.w 1          ;Accessory-Unit
msgbuff:    blk.b 16
            blk.l 128        ;neuer Stack
nstapel:    blk.l 1

contrl:   ;GEM-Parameter-Block
opcode:     blk.w 1
sintin:     blk.w 1
sintout:    blk.w 1
saddrin:    blk.w 1
saddrout:   blk.l 1
            blk.w 5

global:     blk.l 8

intin:      blk.w 80
ptsin:      blk.w 80
intout:     blk.w 80
ptsout:     blk.w 80
addrin:     blk.w 80
addrout:    blk.w 80
```

Dieses Programm wurde auf dem Macro-Assembler SEKA erstellt, wodurch einige Punkte anders aussehen als bei Programmen für den DRI-Assembler, der in dem ATARI-Entwicklungspaket enthalten ist. Zu ändern wären jedoch lediglich die Kommentarzeilen, die für den DRI-Assembler mit einem Sternchen (*) beginnen müssen, und die 'blk'-Anweisung, die bei DRI 'ds' lauten muß.

Das Programm ist in mehrere Teile untergliedert:

1. Anmeldung des Accessories

2. Bereitschafts-Schleife, die im normalen Betrieb des ATARI ST ständig im Hintergrund mitläuft und daher kein Ende haben darf
3. Auswahl-Dialogfenster darstellen und bedienen, dabei die gewählte Kapazität in 'kapazi' ablegen
4. Sicherheits-Abfrage darstellen und bearbeiten
5. vorher benutzten Speicher abmelden (MFREE)
6. neuen Speicher reservieren, ggf. Fehlermeldung ausgeben
7. BIOS-Vektoren für die Disk-Routinen retten und neue Vektoren einstellen
8. Get BPB-Funktion
9. Read/Write-Funktion
10. Media Change-Funktion
11. Datenfelder für Parameterblöcke

Die Punkte 7 bis 10 wurden bereits im vorigen Kapitel besprochen. Die Funktionen der Punkte 1 bis 6 sind für die Beschreibung an dieser Stelle zu umfangreich. Die vollständige Beschreibung der verwendeten Funktionen finden Sie im ATARI ST INTERN- oder im GEM-Buch von DATA BECKER.

Und hier wieder ein BASIC-Lader-Programm, welches das Accessory-Programm RAMDISK.ACC auf der Diskette erstellt:

```
10   '*** Erstellung des RAM-Disk-Accessories ***
15   '
20   ?:fullw 2:clearw 2:gotoxy 0,0
25   ? "File >> RAMDISK.ACC << wird erzeugt":?:?:?
30   dim c%( 741):cs#=0
35   for i=0 to 741
40   read a$:c%(i)=val("&H"+a$)
45   check#=check#+(c%(i))
50   next i
55   if check#= 5104824 then 70
60   ?"Geht leider nicht, da etwas mit den DATAs nicht stimmt."
65   goto 80
70   bsave "RAMDISK.ACC",varptr(c%(0)), 1483
75   ? "Das Programm >> RAMDISK.ACC << ist nun geschrieben."
```

```
80   ?:?:?:?"Bitte Taste drücken":a=inp(2):end
85   '
90   '********* DATAs für RAMDISK.ACC *********
95   '
100 DATA 601A,0000,0546,0000,0000,0000,0000,0000
101 DATA 0000,0000,0000,0000,0000,0000,2E7C,0000
102 DATA 0B60,33FC,000A,0000,0546,33FC,0000,0000
103 DATA 0548,33FC,0001,0000,054A,33FC,0000,0000
104 DATA 054C,33FC,0000,0000,054E,6100,02F6,33F9
105 DATA 0000,06BC,0000,094C,33FC,004D,0000,0546
106 DATA 33FC,0005,0000,054A,33FC,0000,0000,054C
107 DATA 33FC,0000,0000,054E,6100,02C8,33F9,0000
108 DATA 06BC,0000,094A,33FC,0023,0000,0546,33FC
109 DATA 0001,0000,0548,33FC,0001,0000,054A,33FC
110 DATA 0001,0000,054C,33F9,0000,094C,0000,057C
111 DATA 23FC,0000,03F4,0000,07FC,6100,0286,33F9
112 DATA 0000,06BC,0000,094E,6100,0234,0C79,0028
113 DATA 0000,0950,6600,FFF2,3039,0000,0958,B079
114 DATA 0000,094E,6600,FFE2,6100,0006,6000,FFDA
115 DATA 23FC,0000,0402,0000,07FC,6100,02BE,33F9
116 DATA 0000,06BC,0000,0948,0C79,0001,0000,0948
117 DATA 6700,0278,0C79,0003,0000,0948,6700,0044
118 DATA 5479,0000,04BE,0C79,0012,0000,04BE,6D00
119 DATA 0008,4279,0000,04BE,41F9,0000,04C0,4280
120 DATA 3039,0000,04BE,33F0,0800,0000,04BC,E348
121 DATA 41F9,0000,04D2,23F0,0800,0000,042C,6000
122 DATA FF90,23FC,0000,0434,0000,07FC,6100,024C
123 DATA 0C79,0002,0000,06BC,6700,0010,6100,01D2
124 DATA 4A79,0000,04BE,6600,0004,4E75,33FC,0002
125 DATA 0000,0544,4287,3E39,0000,04BC,0687,0000
126 DATA 0009,EB87,EB87,2F07,3F3C,0048,4E41,5C8F
127 DATA 4A80,6700,0018,23C0,0000,04F6,2F3C,0000
128 DATA 01C2,3F3C,0026,4E4E,5C8F,4E75,23FC,0000
129 DATA 046E,0000,07FC,6100,01E2,6000,01AE,23F9
130 DATA 0000,0472,0000,093C,23FC,0000,0258,0000
131 DATA 0472,23F9,0000,0476,0000,0940,23FC,0000
132 DATA 0272,0000,0476,23F9,0000,047E,0000,0944
133 DATA 23FC,0000,02C2,0000,047E,2079,0000,04F6
134 DATA 203C,0000,0900,4298,51C8,FFFC,2079,0000
135 DATA 04F6,D1FC,0000,000B,43F9,0000,0532,7011
```

```
136  DATA 10D9,51C8,FFFC,3E39,0000,04BC,33C7,0000
137  DATA 0520,E34F,0647,0012,2079,0000,04F6,D1FC
138  DATA 0000,0013,10C7,E04F,1087,08F9,0002,0000
139  DATA 04C5,4E75,0C6F,0002,0004,6700,000A,2079
140  DATA 0000,093C,4ED0,203C,0000,0512,4E75,0C6F
141  DATA 0002,000E,6700,000A,2079,0000,0940,4ED0
142  DATA 302F,000C,48C0,E188,E388,206F,0006,322F
143  DATA 000A,5341,2279,0000,04F6,D3C0,302F,0004
144  DATA 0800,0000,6700,0004,C348,203C,0000,01FF
145  DATA 10D9,51C8,FFFC,51C9,FFF2,4240,4E75,0C6F
146  DATA 0002,0004,6700,000A,2079,0000,0944,4ED0
147  DATA 3039,0000,0544,4279,0000,0544,4E75,33FC
148  DATA 0019,0000,0546,33FC,0010,0000,0548,33FC
149  DATA 0007,0000,054A,33FC,0001,0000,054C,23FC
150  DATA 0000,0950,0000,07FC,43F9,0000,03D4,45F9
151  DATA 0000,057C,700F,34D9,51C8,FFFC,6100,0004
152  DATA 4E75,223C,0000,04FA,303C,00C8,4E42,4E75
153  DATA 4AB9,0000,04F6,6700,0032,2F3C,0000,0376
154  DATA 3F3C,0026,4E4E,5C8F,2F39,0000,04F6,3F3C
155  DATA 0049,4E41,5C8F,4A80,6700,0010,23FC,0000
156  DATA 0494,0000,07FC,6100,0032,42B9,0000,04F6
157  DATA 4E75,23F9,0000,093C,0000,0472,23F9,0000
158  DATA 0940,0000,0476,23F9,0000,0944,0000,047E
159  DATA 08B9,0002,0000,04C5,4E75,33FC,0034,0000
160  DATA 0546,33FC,0001,0000,0548,33FC,0001,0000
161  DATA 054A,33FC,0001,0000,054C,33FC,0000,0000
162  DATA 054E,33FC,0001,0000,057C,6100,FF56,4E75
163  DATA 0013,0001,0001,0001,0000,0000,0000,0000
164  DATA 0000,0000,0000,0000,0000,0000,0000,0000
165  DATA 2020,5241,4D2D,4469,736B,2043,0000,5B31
166  DATA 5D5B,5769,6576,6965,6C20,5241,4D2D,4469
167  DATA 736B,2D4B,4279,7465,203F,5D5B,4578,6974
168  DATA 7C20,6D65,6872,207C,2031,3030,205D,0000
169  DATA 5B31,5D5B,416C,7465,6E20,496E,6861,6C74
170  DATA 2064,6572,7C20,5241,4D2D,4469,736B,206C
171  DATA 9473,6368,656E,203F,5D5B,204A,6120,2120
172  DATA 7C20,4E65,696E,205D,0000,5B32,5D5B,4E69
173  DATA 6368,7420,6765,6E75,6720,5241,4D20,215D
174  DATA 5B44,616E,6E20,6E69,6368,742E,2E5D,0000
175  DATA 5B32,5D5B,4665,686C,6572,2062,6569,204D
```

```
176 DATA 4652,4545,2021,5D5B,4461,6E6E,206E,6963
177 DATA 6874,2E2E,5D00,0000,0064,0002,0000,0064
178 DATA 00C8,012C,0190,01F4,0258,02BC,0320,2020
179 DATA 3020,2031,3030,2032,3030,2033,3030,2034
180 DATA 3030,2035,3030,2036,3030,2037,3030,2038
181 DATA 3030,0000,0000,0000,0546,0000,055C,0000
182 DATA 057C,0000,06BC,0000,07FC,0000,089C,0200
183 DATA 0002,0400,0007,0005,0006,0012,FFFF,0000
184 DATA 0000,0000,0000,0000,0000,0000,0000,0002
185 DATA 0201,0002,7000,FFFF,0005,0009,0001,0000
186 DATA 0000,0000,0002,0808,0808,080A,0408,0808
187 DATA 080A,0408,0808,0806,0406,040A,040C,0A06
188 DATA 1204,0A04,080C,0A08,0A06,0808,0808,0A04
189 DATA 0C0E,0E08,2006,1004,1206,0E06,0E06,0A12
190 DATA 0C0E,060C,2608,121C,3408,060A,0808,0806
191 DATA 0406,0614,0E0A,0E14,040A,080A,0A16,0808
192 DATA 0808,0801,3204,0404,0404,0000
```

Wenn Sie nun Ihren Rechner eingeschaltet und die RAM-Disk installiert haben, müssen Sie oft etliche Programme auf diese RAM-Disk kopieren, bevor Sie arbeiten können. Diese ewige Kopiererei ist nicht nur lästig, sondern auch zeitaufwendig. Dieses Problem kann jedoch auf einfache Weise gelöst werden.

## 6.2 Disk-to-RAM-Disk Copy

Das folgende Programm kopiert einfach den gesamten Inhalt einer einseitigen Diskette in die RAM-Disk C. Hierbei werden alle Sektoren von 0 (logische Sektornummer) bis 9*80-1, also 719, von dem gewählten Laufwerk gelesen und in die 'Sektoren' der RAM-Disk kopiert. Dabei ist natürlich zu beachten, daß die Kapazität der RAM-Disk mindestens 400 KByte betragen muß, damit der Sektor 719 auch existiert.

Um das Programm so schnell wie möglich zu machen, werden pro Aufruf der Lese- bzw. Schreiberoutine 'FLOPRW' jeweils 9 Sektoren hintereinander gelesen. Den leichten Geschwindigkeitsvorteil gegenüber dem einzelnen Kopieren von Sektoren haben

wir dem DMA-Chip zu verdanken, der mit einer Programmierung alle 9 Sektoren (jeweils ein ganzer Track) auf einmal liest und zum Rechner zurückschickt. Das ergibt zugegebenermaßen kein umwerfendes Geschwindigkeits-Plus, aber immerhin ein wenig. Noch schneller ginge es natürlich, wenn gleich alle Sektoren der Diskette auf einmal gelesen würden, doch gibt dies evtl. Speicherplatzprobleme.

Legt man eine zweiseitige Diskette in das zu lesende Laufwerk ein, so erscheinen im Inhaltsverzeichnis der RAM-Disk natürlich alle Dateinamen des Originals. Das Inhaltsverzeichnis ist ja auch vollständig kopiert worden, nicht jedoch die hintere Hälfte der Sektoren. Ist die Originaldiskette über die Hälfte belegt, so lassen sich die dort liegenden Programme und Dateien nicht von der RAM-Disk laden. Ansonsten funktioniert das Programm auch mit zweiseitigen Disketten einwandfrei.

Sehen wir uns nun das Programm an:

```
;*** Disk - to - RAM-Disk - Copy S.D. ***

run:
        clr.l   ap1rsv
        clr.l   ap2rsv
        clr.l   ap3rsv
        clr.l   ap4rsv
        move    #10,opcode          ;appl_init
        move    #0,sintin
        move    #1,sintout
        move    #0,saddrin
        move    #0,sintin
        jsr     aes
        move    #77,opcode          ;graf_handle
        move    #5,sintout
        move    #0,saddrin
        move    #0,saddrout
        jsr     aes
        move    intout,grhandle
```

```
        move.l   #alarm,d0
        bsr      formalert           ;Auswahlfenster ausgeben
        subq     #2,d0               ;Laufwerksnummer korrigieren
        tst      d0
        bmi      quit                ;Abbruch

        move     d0,drive            ;Laufwerksnummer retten
        clr      sector              ;Beginn bei Sektor 0

loop:
        move     drive,d1            ;gewählte Diskette
        move     #2,d0               ;Read
        bsr      floprw              ;9 Sektoren lesen
        bne      readerr             ;Fehler beim Lesen !
        move     #2,d1               ;Drive C = RAM-Disk
        move     #1,d0               ;Write
        bsr      floprw              ;9 Sektoren schreiben
        bne      wrerr               ;Fehler beim Schreiben !
        add      #9,sector           ;Sektornummer + 9
        cmp      #9*80,sector        ;Ende ?
        blt      loop                ;nein

quit:
        clr      -(sp)
        trap     #1                  ;Exit => Desktop

floprw: ;Read/Write Diskette
        move     d1,-(sp)            ;Laufwerk
        move     sector,-(sp)        ;Start-Sektor
        move     #9,-(sp)            ;9 Sektoren lesen/schreiben
        pea      puffer              ;Puffer
        move     d0,-(sp)            ;Read/Write
        move     #4,-(sp)
        trap     #13                 ;rwabs-Funktion
        add.l    #14,sp
        tst      d0                  ;Test auf Fehler
        rts

readerr:
        move.l   #reer,d0
```

```
        bsr         formalert              ;"Fehler beim Lesen !"
        bra         quit

wrerr:
        move.l      #wrer,d0
        bsr         formalert              ;"Fehler beim Schreiben"
        bra         quit

aes: ;AES-Aufruf
        move.l      #aespb,d1
        move        #$c8,d0
        trap        #2
        rts

formalert:
        move        #52,contrl             ;form_alert
        move        #1,contrl+2
        move        #1,contrl+4
        move        #1,contrl+6
        move        #0,contrl+8
        move        #1,intin
        move.l      d0,addrin
        jsr         aes
        move        intout,d0
        rts

alarm:   dc.b  "[1][Von welchem Laufwerk| kopieren ?]"
         dc.b  "[Exit| A | B ]",0,0
reer:    dc.b  "[2][Fehler beim Lesen !][Quit]",0,0
wrer:    dc.b  "[2][Fehler beim Schreiben !][Quit]",0,0

even
aespb:   dc.l  contrl,global,intin,intout,addrin,addrout

data
contrl:    ;diverse Felder für das AES
opcode:    blk.w 1
sintin:    blk.w 1
sintout:   blk.w 1
saddrin:   blk.w 1
```

```
saddrout: blk.l 1
          blk.w 5

global:   blk.w 7
ap1rsv:   blk.l 1
ap2rsv:   blk.l 1
ap3rsv:   blk.l 1
ap4rsv:   blk.l 1

intin:    blk.w 128
ptsin:    blk.w 128
intout:   blk.w 128
ptsout:   blk.w 128
addrin:   blk.w 128
addrout:  blk.w 128

grhandle: blk.w 1
drive:    blk.w 1                      ;Laufwerks-Nummer
sector:   blk.w 1                      ;Sektoren-Zähler
puffer:   blk.b 9*512                  ;Puffer für 9 Sektoren
```

Durch den recht einfachen Aufbau des Programmes sind einige Variationen im Programm leicht möglich. So können Sie z.B. durch Änderung der Ende-Bedingung im CMP #9*80,SECTOR -Befehl auch doppelseitige Disketten in eine 800 KByte-RAM-Disk kopieren, wenn Sie dort einfach #9*80*2 einsetzen.

Eine weitere denkbare Variation wäre es, die Kopierrichtung ebenfalls wählbar zu machen. Dadurch wäre es möglich, nach der Arbeit in der RAM-Disk seine Arbeit auf Diskette zurückzuspeichern.

Auch noch interessant wäre es, das Programm in ein Accessory umzuwandeln. Ausgestattet mit allen möglichen Zusatzfunktionen könnte dies ein recht brauchbares Werkzeug ergeben.

Doch nun auch den BASIC-Lader. Er erstellt das Programm DTRCOPY.PRG auf der Diskette:

```
10    '*** Erstellung des Disk-to RAM-Disk-Copy ***
15    '
20    ?:fullw 2:clearw 2:gotoxy 0,0
25    ? "File >> DTRCOPY.PRG << wird erzeugt":?:?:?
30    dim c%( 286):cs#=0
35    for i=0 to 286
40    read a$:c%(i)=val("&H"+a$)
45    check#=check#+(c%(i))
50    next i
55    if check#= 2531110 then 70
60    ?"Geht leider nicht, da etwas mit den DATAs nicht stimmt."
65    goto 80
70    bsave "DTRCOPY.PRG",varptr(c%(0)), 573
75    ? "Das Programm >> DTRCOPY.PRG << ist nun geschrieben."
80    ?:?:?:?"Bitte Taste drücken":a=inp(2):end
85    '
90    '********* DATAs für DTRCOPY.PRG *********
95    '
100   DATA 601A,0000,01F2,0000,0000,0000,0000,0000
101   DATA 0000,0000,0000,0000,0000,0000,42B9,0000
102   DATA 0216,42B9,0000,021A,42B9,0000,021E,42B9
103   DATA 0000,0222,33FC,000A,0000,01F2,33FC,0000
104   DATA 0000,01F4,33FC,0001,0000,01F6,33FC,0000
105   DATA 0000,01F8,33FC,0000,0000,01F4,4EB9,0000
106   DATA 010E,33FC,004D,0000,01F2,33FC,0005,0000
107   DATA 01F6,33FC,0000,0000,01F8,33FC,0000,0000
108   DATA 01FA,4EB9,0000,010E,33F9,0000,0426,0000
109   DATA 0826,203C,0000,0160,6100,009E,5540,4A40
110   DATA 6B00,0044,33C0,0000,0828,4279,0000,082A
111   DATA 3239,0000,0828,303C,0002,6100,002E,6600
112   DATA 004E,323C,0002,303C,0001,6100,001E,6600
113   DATA 004C,0679,0009,0000,082A,0C79,02D0,0000
114   DATA 082A,6D00,FFCC,4267,4E41,3F01,3F39,0000
115   DATA 082A,3F3C,0009,4879,0000,082C,3F00,3F3C
116   DATA 0004,4E4D,DFFC,0000,000E,4A40,4E75,203C
117   DATA 0000,0195,6100,0022,6000,FFCC,203C,0000
118   DATA 01B5,6100,0014,6000,FFBE,223C,0000,01DA
119   DATA 303C,00C8,4E42,4E75,33FC,0034,0000,01F2
120   DATA 33FC,0001,0000,01F4,33FC,0001,0000,01F6
121   DATA 33FC,0001,0000,01F8,33FC,0000,0000,01FA
```

```
122  DATA 33FC,0001,0000,0226,23C0,0000,0626,4EB9
123  DATA 0000,010E,3039,0000,0426,4E75,5B31,5D5B
124  DATA 566F,6E20,7765,6C63,6865,6D20,4C61,7566
125  DATA 7765,726B,7C20,6B6F,7069,6572,656E,203F
126  DATA 5D5B,4578,6974,7C20,4120,7C20,4220,5D00
127  DATA 005B,325D,5B46,6568,6C65,7220,6265,696D
128  DATA 204C,6573,656E,2021,5D5B,5175,6974,5D00
129  DATA 005B,325D,5B46,6568,6C65,7220,6265,696D
130  DATA 2053,6368,7265,6962,656E,2021,5D5B,5175
131  DATA 6974,5D00,0000,0000,01F2,0000,0208,0000
132  DATA 0226,0000,0426,0000,0626,0000,0726,0000
133  DATA 0002,0606,0608,0808,0808,0608,0808,0806
134  DATA 0604,0612,0606,2408,100A,180E,0E10,0808
135  DATA 0808,0806,0606,8004,0404,0404,0000
```

# 7. Programmieren in Maschinensprache am Beispiel eines Disk-Monitors

Mit den bisher in diesem Buch angebotenen Programmen können Sie schon einige Daten der Diskette ansehen und ändern, doch was ist ein Floppy-Buch ohne einen Disk-Editor, mit dem man alle Daten der Diskette ändern und ansehen kann? Da ich mittlerweile schon auf eine siebenjährige Erfahrung im Abtippen von Programmen aus Zeitungen und Büchern habe, möchte ich Ihnen den großen Frust ersparen und einen quasi-modularen Aufbau des Programms anbieten, der schon mit relativ wenig Tipparbeit zum Erfolgserlebnis führt.

Das Listing edit.s enthält zwar alle Menüpunkte und alle Daten und Unterprogramme des gesammten Diskeditors, doch wurden nur die Unterroutinen des Sektor-Menüs ausformuliert, alle anderen Routinen enthalten nur ein rts, kehren also bei Aufruf sofort zurück. Im Listing subrout.s folgen dann die ausformulierten Unterprogramme mit gleichem Namen, die Sie dann nach Bedarf an Stelle der Platzhalter ins Programm edit.s einfügen und dieses so bis zur maximalen Größe ausbauen können.

Beim Eingeben des Programms werden Sie schon auf das "größte" Problem bei der Entstehung des Editors stoßen: die Namensgebung der Labels und Variablen. Acht signifikante Buchstaben sind bei einem solch umfangreichen Assemblerprojekt einfach zu wenig, um einprägsame und logische Namen für Unterprogramme und Variablen zu formulieren.

Wenn Sie den Editor sofort nutzen wollen, besteht natürlich auch die Möglichkeit, die Diskette zum Buch beim Verlag zu bestellen, die dann alle Programme und Sources des Floppybuches enthält.

## 7.1 Die TOS-Funktionen zum Floppy-Zugriff

Die Funktionen des Editors bauen zum größten Teil auf Funktionen des Betriebssystems (TOS oder GEMDOS) auf, nur ein kleiner Teil greift direkt auf DMA-Chip und Controller zu. Der Zugriff auf Disk-Controller und DMA-Chip des ATARI ST vom Betriebssystem aus ist von verschiedenen Ebenen des hierarchisch gegliederten TOS möglich.

Die "Hochsprachen" wie Pascal, C, Fortran und BASIC ermöglichen das Arbeiten mit sequentiellen und wahlfreien (RANDOM-ACCESS) Dateien, d.h eine Hochsprache unterteilt die Diskette nicht in Spuren und Sektoren, sondern der Zugriff auf die Diskette ist File-orientiert und bewegt sich nur in den Grenzen dieser Files.

Steigt man einen Schritt tiefer in der Hierarchie des Betriebssystems, so erkennt man, daß sich die Hochsprachen der GEMDOS-Funktionen des ATARI ST bedienen. Diese GEMDOS-Funktionen stellen die Betriebssystemeigenen Routinen für die Hochsprache-Kommandos zur Verfügung, d.h. diese GEMDOS-Routinen sind immer noch File-orientiert und bieten nur wahlfreien und sequentiellen Zugriff auf Files der Diskette.

Beim nächsten Schritt stößt man auf die BIOS-Funktionen, die praktisch den ersten physikalischen Kontakt mit der Diskette ermöglichen, indem diese in logische Sektoren von 0 bis zum Fassungsvermögen (bei DS-Disks 1440, bei SS-Disks 720) eingeteilt wird. Die BIOS-Funktionen erlauben so den Zugriff auf alle Sektoren der Diskette, man weiß allerdings nicht, auf welcher Spur und Seite sich der gelesene logische Sektor befindet. Es ist allerdings möglich, aus den Daten des Bios-Parameterblockes auf diese zu schließen (s.u.). Bedient man sich schließlich der XBIOS-Funktionen, wird der Zugriff auf Track, Seite

und physikalischen Sektor möglich. Der Anwender muß allerdings dabei wissen, wieviele Sektoren sich auf einem Track befinden ect. Dafür bieten die XBIOS-Funktionen die Möglichkeit, solche diskspezifischen Eigenschaften zu bestimmen. So kann man z.B. einzelne Tracks formatieren und dabei die gewünschte Anzahl der Sektoren pro Track angeben.

Als Mittelpunkt aller Routinen erweisen sich schließlich der Disk-Controller WD1772 und der DMA-Chip, die im I/O Bereich des ATARI-Adreßbereiches von $FF800 bis $FFFFF einige Adressen belegen, über die dann sämtliche Funktionen dieser beiden Chips gesteuert werden können.

Zur Verdeutlichung: der BASIC-Befehl WRITE#, der Daten in eine sequentielle Datei schreibt, bedient sich der GEMDOS-Funktion WRITE, welche ihrerseits die BIOS-Funktion RWABS aufruft, wobei sich diese die XBIOS-Funktion FLOPWR zunutze macht um letztlich DMA- und Controller-Chip mitzuteilen, wohin welche Daten auf Diskette geschrieben werden müssen.

Alle Betriebsystems-Funktionen (GEMDOS, BIOS, XBIOS) werden im DATA BECKER Buch "ATARI ST INTERN" ausführlich beschrieben, so daß ich hier nur auf die insgesamt acht BIOS und XBIOS Aufrufe eingehe, die der direkten Kommunikation mit der Diskette dienen. Alle Aufrufe erwarten zu übergebende Parameter auf dem Stack und geben eventuelle Ergebnisse oder im Fehlerfall einen negativen Fehlercode im Register D0 zurück. Nach einem Aufruf sind in den meisten Fällen die Register D0-D2 und A0-A2 verändert, müssen folglich bei Bedarf "gerettet" werden. Die beiden BIOS-Funktionen RWABS und GETBPB werden über den BIOS-spezifischen TRAP #13 aufgerufen und bewirken folgendes:

Rwabs: BIOS-Funktionsnummer 4

Diese sehr flexible Funktion dient sowohl zum Lesen als auch zum Schreiben von einem oder mehreren logischen Sektoren. Diese Sektoren können sowohl auf der physikalischen Diskette

als auch auf der Harddisk und schließlich in einer RAM-Disk liegen. Als Parameter werden übergeben:

device: bestimmt das Laufwerk auf welches zugegriffen wird. Die Nummerierung fängt bei 0 für Laufwerk A an und ist noch oben unbegrenzt. Die in diesem Buch im Kapitel 6.1 vorgestellte RAM-Disk, die als Laufwerk C angesprochen wird, hat also eine device-Nummer von 2.

recnr: gibt die logische Nummer des zu bearbeitenden Sektors an. Zählbeginn ist wiederum die 0. Die maximale Anzahl der Sektoren schwankt je nach Gerät: so passen zum Beispiel auf eine einseitige 80 Track-Diskette im ATARI Format (double Density) 720 logische Sektoren, von denen aber für User-Daten "nur" 702 zur Verfügung stehen. Auf den achtzehn übrigen Sektoren verwaltet das TOS die Userdaten mittels Directory und FAT (File Allocation Table).

anzahl: bestimmt die Anzahl der zu bearbeitenden logischen Sektoren.

puffer: ist eine Adresse, aus der bzw. in die die Daten der Floppy geschrieben werden sollen. Möchten Sie also 4 logische Sektoren des ATARI-spezifischen Formats (512 Bytes/Sektor) lesen, müssen ab Adresse 'puffer' 4*512=2048 freie Bytes zur Verfügung stehen.

rwflag: schließlich bestimmt, ob gelesen oder geschrieben wird. Es sind 4 verschiedene Werte möglich:

| wflag: | Bedeutung: |
|---|---|
| 0 | Sektoren lesen |
| 1 | Sektoren schreiben |
| 2 | Sektoren unbedingt lesen (auch bei Diskwechsel) |
| 3 | Sektoren unbedingt schreiben " |

Ein möglicher Aufruf in Maschinensprache könnte so aussehen:

```
move.w #0,-(a7)          * Laufwerk A (device)
move.w #11,-(a7)         * recnr Beginn bei log. Sekt. 11
move.w #5,-(a7)          * anzahl,alle 5 Directory S
move.l #puffer,-(a7)     * Adresse des freien Platzes
move.w #2,-(a7)          * rwflag, unbedingt lesen
move.w #4,-(a7)          * BIOS Funktionsnummer
trap   #13               * BIOS Aufruf
add.l  #14,a7            * Stack restaurieren
tst.w  d0                * prüfen ob ein Fehler aufgetreten ist
bmi    error             * negativer Wert bedeutet Fehler
...
...                      * hier geht es im Normalfall weiter, die
...                      * gelesenen Daten befinden sich jetzt im
...                      * RAM ab Adresse 'puffer'
```

Getbpb BIOS Funktionsnummer 7

Der BIOS-Parameterblock beeinhaltet die Daten der aktuellen Diskette. Diese Daten befinden sich im Bootsektor der Diskette und werden beim Aufruf dieser Funktion in den sich im RAM befindenden BPB (Biosparameterblock) eingetragen. Aufruf von Assembler:

```
move.w  device,-(a7)  * Nummer des Laufwerkes 0 = A
move.w  #7,-(a7)      * BIOS Nummer
trap    #13
addq.l  #4,a7         * Stack bereinigen
tst.w   d0
bmi     error         * Im Fehlerfall ist D0 negativ
...
...
...                   * sonst steht in D0 die Adresse des BPB.
...                   * Im Normalfall erhält man $4DCE für
...                   * Laufwerk A und $4DEE für Laufwerk B
```

Ab der in D0 zurückgebenen Adresse befinden sich die Daten in Wortgröße (2 Byte) im Speicher, als da wären:

*Laufwerk A*

| Adresse: | Name: | Bedeutung | SS | DS |
|---|---|---|---|---|
| $4DCE | recsiz | Sektorgröße in Bytes | 512 | 512 |
| $4DD0 | clsiz | Clustergröße in Sektoren | 2 | 2 |
| $4DD2 | clsizb | Clustergröße in Bytes | 1024 | 1024 |
| $4DD4 | rdlen | Directorylänge in Sektoren | 7 | 7 |
| $4DD6 | fsiz | FAT-Größe in Sektoren | 5 | 5 |
| $4DD8 | fatrec | Sektornummer des zweite FAT | 6 | 6 |
| $4DDA | datrec | Sektornummer des ersten Datencl. | 18 | 18 |
| $4DDC | numcl | Anzahl der Cluster auf Disk | 351 | 711 |
| $4DDE | bflags | diverse Flags | | |
| $4DE0 | unbekannt | | | |
| $4DE2 | nside | Anzahl der Seiten der Disk | 1 | 2 |

Die Beispieldaten gelten jeweils für das ATARI-spezifische Aufzeichnungsformat mit 80 Tracks (SS = einseitig formatierte Disketten, DS = doppelseitig formatierte Disketten).

Mediach BIOS Funktionsnummer 9

Diese Funktion prüft anhand des Diskettennamens, ob zwischenzeitlich die Diskette gewechselt wurde. Als Parameter wird die Laufwerksnummer übergeben.

```
move.w   device,-(a7)          * Laufwerksnummer (z.B. 0 für A)
move.w   #9,-(a7)              * BIOS Funktionsnummer
trap     #13                   * Aufruf
addq.l   #4,a7                 * Stack restaurieren
...
```

Die in D0 zurückgegeben Werte liegen zwischen 0 und 2 und bedeuten:

| Nummer: | Bedeutung: |
|---|---|
| 0 | Diskette wurde nicht gewechselt |
| 1 | Diskette wurde möglicherweise gewechselt |
| 2 | Diskette wurde gewechselt |

Hier folgen nun die vier für unsere Zwecke wichtigen XBIOS Funktionen:

Floprd Funktionsnummer 8

Mit dieser Funktion kann man einen oder mehrere hintereinanderliegende Sektoren eines Tracks lesen. Die zu übergebenden Parameter sind:

anzahl: bestimmt, wie viele Sektoren gelesen werden sollen. Die bei den ATARI-Disketten möglichen Werte schwanken zwischen 1 und 10. Zehn Sektoren können allerdings nur gelesen werden, wenn sie mittels eines speziellen Formatierprogrammes auch auf der Diskette existieren, da das ATARI- eigene Formatierprogramm "nur" neun Sektoren pro Track auf die Diskette schreibt.

seite: gibt die Seite der Diskette an, 0 oder 1.

track: bestimmt den Track, auf dem sich der zu lesende Sektor befindet.

sector: nun der physikalische Sektor selbst

device: der schon bekannte Laufwerksparameter (0=A)

filler: ein Langwort, das keine Bedeutung hat und wohl für Erweiterungen der Funktion gedacht ist

puffer: die Adresse, an die die Diskdaten übertragen werden sollen

```
        move.w  #1,-(a7)          * anzahl, einen Sektor
        move.w  #0,-(a7)          * seite
        move.w  #0,-(a7)          * Track null
        move.w  #1,-(a7)          * Sektor eins = Bootsektor
        move.w  #0,-(a7)          * Laufwerk A
        move.l  #0,-(a7)          * filler, dummy Langwort
        move.l  #puffer,-(a7)     * Adresse des Datenzielortes
        move.w  #9,-(a7)          * XBIOS Funktionsnummer
        trap    #14
        add.l   #20,a7            *
        tst.w   d0
        bmi     error
        ...
puffer: ds.b 512
```

Flopwr: XBIOS Funktionsnummer 9

Analog zur vorherigen Funktion können mit Flopwr Sektoren auf Diskette geschrieben werden. Die zu übergebenden Parameter sind die gleichen wie bei Floprd.

```
        move.w  #4,-(a7)          * anzahl, vier Sektoren
        move.w  #0,-(a7)          * seite
        move.w  #5,-(a7)          * Track fünf
        move.w  #1,-(a7)          * Sektor 1 = Startsekt. des
                                  * Schreibens
        move.w  #0,-(a7)          * Laufwerk A
        move.l  #0,-(a7)          * filler, dummy Langwort
        move.l  #puffer,-(a7)     * Adresse der zu schreib. Daten
        move.w  #9,-(a7)          * XBIOS Funktionsnummer
        trap    #14
        add.l   #20,a7            *
```

```
        tst.w  d0
        bmi    error
        ...
puffer: ds.b   4       * 512
```

Dieser Aufruf schreibt die 2048 Datenbytes, die ab Adresse 'puffer' im Speicher stehen, auf die Sektoren 1,2,3,4 des Tracks 5 auf Seite 0 der Diskette.

Flopfmt: XBIOS Funktionsnummer 10: Diese Routine ermöglicht das Formatieren eines Tracks mit 1-10 Sektoren pro Track. Die Parameter sind:

virgin: Dieses Wort bestimmt die neuen Sektorinhalte, also die Daten, die in die einzelnen Sektoren eingetragen werden. Es empfiehlt sich, den gleichen Wert wie das TOS ($E5E5) zu nehmen, da Bytewerte größer $EF unter Umständen nicht als Byte geschrieben werden, sondern Sonderfunktionen darstellen und z.B. Adressmarks oder Checkssums der bisher geschriebenen Bytes schreiben (sehen sie dazu doch bitte ins Kapitel über den Floppy Disk Controller).

magic: Dies ist die magische Konstante $87654321.

interleave: bestimmt den Sektorversatz auf der Diskette. Computer mit wenig intelligentem Floppydiskcontroller ohne DMA müssen die von der Diskette gelesenen Daten noch auswerten, was ja etwas Zeit kostet. Dadurch kann es vorkommen, daß der nächste Sektor schon am Schreiblesekopf vorbeirotiert ist, wenn die CPU ihre Arbeit beendet hat. Schreibt man nun die Sektoren nicht in der Reihenolge 1 2 3 4 5 6 7 8 9 (interleave = 1), sondern z.B. 1 6 2 7 3 8 4 9 5 (interleave = 2) auf die Diskette, so kann die Zeit zwischen dem aktuellen und dem nächsten Sektor für die Datenauswertung ausreichen. Dies

kann die Datenübertragung erheblich beschleunigen, da sonst die Diskette erst eine ganze Umdrehung bis zum Auffinden des Folgesektors zurücklegen müßte. Beim ATARI ST wird die Datenauswertung vom Controller vorgenommen, so daß die Sektoren ohne Versatz geschrieben werden können. Ein Wert von eins sollte also hier übergeben werden.

seite: die Diskettenseite

track: der Zieltrack

spt: Anzahl der Sektoren pro Track. Die Diskette "verkraftet" zehn Sektoren pro Track, das TOS arbeitet mit neun.

device: Laufwerksnummer

filler: wieder ein dummy-Langwort für spätere Erweiterungen

puffer: bestimmt die Adresse, an der das XBIOS den kompletten Track aufbaut (mit allen Sync-Bytes und Adreßfeldern). Es werden ca. 8 KByte benötigt.

Funktionsnummer ist 10

```
move.w  #$e5e5,-(a7)        * virgin
move.l  #$87654321,-(a7)    * magic
move.w  #1,-(a7)            * interleave gleich 1
move.w  #0,-(a7)            * Seite 0
move.w  #5,-(a7)            * Track 5
move.w  #9,-(a7)            * spt, 9 Sektoren pro Track
move.w  #0,-(a7)            * device 0 gleich Laufwerk A
move.l  #0,-(a7)            * filler, dummy Langwort
move.l  #puffer,-(a7)       * Adresse des freien Speichers
move.w  #10,-(a7)           * XBIOS Funktionsnummer
```

```
        trap    #14
        add.l   #26,a7
        tst.w   d0              * Fehler aufgetreten ?
        bmi     error           * ja
...
...
puffer: ds.b    8*1024          * hier ist Platz
```

*Protobt: XBIOS Funktionsnummer 18*

Diese Funktion erleichtert das Erstellen eines Bootsektors für verschiedene Diskettenformate. Zuerst liest man den Sektor 1 von Track 0, Seite 0 einer beliebigen formatierten Diskette ein, ruft anschließend Protobt auf und schreibt nun den durch Protobt geänderten Bootsektor wieder auf Sektor 1, Track 0, Seite 0 der Diskette, auf der der Bootsektor erzeugt werden soll. Die zu übergebenen Parameter:

execflag: gibt an, ob der Bootsektor ausführbar ist, d.h. ob sich ab Byte 30 relativ zum Sektoranfang ein ausführbares Bootprogramm befindet. Mögliche Werte sind:

| Wert: | Bedeutung: |
|---|---|
| 0 | nicht ausführbar |
| 1 | ausführbar |
| -1 | Puffer bleibt so wie er war |

disktyp: Diskettentyp

0 40 Track, single sided (SS, SD 180 K-Byte)
1 40 Track, double sided (DS, SD 360 K-Byte)
2 80 Track, single sided (SS, DD 360 K-Byte)
3 80 Track, double sided (DS, DD 720 K-Byte)
-1 Disktyp bleibt unverändert

Die ATARI Formate sind, wie Sie sicher schon erkannt haben, die double Density Formate 2 und 3.

serialnr: die Serienummer bedeutet eine 24-Bit große Zahl, die in den Bootsektor geschrieben wird und an der das Betriebssystem einen eventuellen Diskettenwechsel erkennt. Ist die übergebene Seriennummer größer als 24-Bit (z.B. $01000000), so schreibt das Betriebssystem eine Zufallszahl, ist sie -1, wird die Pufferseriennummer nicht verändert.

puffer: bedeutet die Adresse, an der sich der zu verändernde Bootsektor befindet (512 Bytes Platz).

```
        move.w  #-1,-(a7)       * execflag, Ausführbarkeit nicht ändern
        move.w  #3,-(a7)        * disktyp, 80 Track,
        move.l  #$04000000,-(a7)* serialnr, Zufallsseriennummer
        move.l  #puffer,-(a7)   * Hier befindet sich der Sektor
        move.w  #18,-(a7)       * XBIOS Funktionsnummer
        trap    #14,-(a7)
        add.l   #14,a7
        tst.w   d0
        bmi     error           * hat nicht geklappt
        ...
        ...
puffer: ds.b    512
```

Dieses Programmfragment wandelt einen von einer einseitigen Bootdiskette eingelesenen Bootsektor in einen solchen für eine zweiseitige Diskette um, der dann auf die formatierte zweiseitige Diskette geschrieben werden kann. So kann auch von einer zweiseitigen Diskette das Betriebssystem geladen werden.

Nun schauen wir uns einmal an, welche Informationen sich im Bootsektor befinden.

Die 16-Bit Daten stehen im Intel-Format (erst Low-Byte, dann High-Byte) auf der Diskette, und ein ausführbarer Bootsektor ist durch eine Checksum von $1234 gekennzeichnet.

| Byte | | 40 Track SS | 40 Tr DS | 80 Tr SS | 80 Tr DS |
|---|---|---|---|---|---|
| 0,1 | bra 30 | Sprung nach $30 sofern der Bootsektor ausführbar ist | | | |
| 2-7 | Text: 'Loader' | | | | |
| 8-10 | serialnr | | | | |
| 11-12 | bps | 512 | 512 | 512 | 512 |
| 13 | spc | 2 | 2 | 2 | 2 |
| 14-15 | res | 1 | 1 | 1 | 1 |
| 16 | fat | 2 | 2 | 2 | 2 |
| 17-18 | dir | 64 | 112 | 112 | 112 |
| 19-20 | sec | 360 | 720 | 720 | 1440 |
| 21 | media | 252 | 253 | 248 | 249 |
| 22-23 | spf | 2 | 2 | 5 | 5 |
| 24-25 | spt | 9 | 9 | 9 | 9 |
| 26-27 | side | 1 | 2 | 1 | 2 |
| 28-29 | hide | 0 | 0 | 0 | 0 |
| 30 | bootcode: | ab hier steht bei einem ausführbaren Bootsektor der Bootcode | | | |
| .. | | | | | |
| .. | | | | | |
| 510-511 | Checksumme des gesamten Bootsektors von Byte 0 bis 509 | | | | |

## 7.2. Das Listing und die Bedienung des Disk-Editors

Wie weiter oben schon erwähnt, folgt hier nun der erste Teil des Disk-Editors. Dieses Listing ist für den Assembler von Digital Research (Entwicklungspaket) vorgesehen und mit diesem so zu verarbeiten. Nachdem Sie das Listing richtig abgetippt und assembliert haben funktioniert in dem dann vorhandenen Programm "edit.tos" nur das Sektormenü. Zum Vollausbau zum gesamten Programms müssen Sie die "dummy-Unterprogramme" noch durch die vollwertigen Unterprogramme ersetzen. Der BASIC-Lader für das komplette Programm befindet sich im Anhang I.

```
**********************************************************************
*                                                                    *
* The little Diskeditor, U. Braun , August 1986                      *
*                                                                    *
* DATA BECKER FLOPPY-BUCH                                            *
*                                                                    *
**********************************************************************


        text

**********************************************************************
*                                                                    *
* Einsprung nach dem Laden, Länge berechnen, und Platz reservieren   *
*                                                                    *
**********************************************************************

sstart: move.l a7,a5      * auf dem Stack ist die Basepageadresse
        move.l 4(a5),a5   * basepage address = Programmanfang - $100
        move.l $c(a5),d0  * Programmlänge
        add.l  $14(a5),d0* Länge des initialisierten Datenbereichs
        add.l  $1c(a5),d0* Länge des nicht initialisierten Datenbereichs
        add.l  #$1100,d0  * 4 K-Byte Userstack=reichlich Platz
        move.l a5,d1      * Startadresse des Programms
        add.l  d0,d1      * Plus Anzahl belegter Bytes = Platzbedarf
        and.l  #-2,d1     * gerade Adresse für Stack
        move.l d1,a7      * Userstackpointer auf letzten 4K- Byte
        move.l d0,-(sp)   * Länge des reservierten Bereichs
        move.l a5,-(sp)   * Anfangsadresse des reservierten Bereichs
        move.w d0,-(sp)   * Dummy-Wort
        move.w #$4a,-(sp)* GEM Dos Funktion SETBLOCK
        trap   #1
        add.l  #12,sp     * alte Stackadresse wieder restaurieren
        move.w #3,-(a7)   * Repeat-Rate ändern wegen Überlauf
        move.w #$b,-(a7)  * des Tastaturpuffers bei schnellerer
        move.w #35,-(a7)  * Repeat-Rate
        trap   #14
        addq.l #6,a7
        jsr    haupt      * Sprung zum Hauptprogramm . ( User-erstellt )
        move.l #0,-(a7)   * Beendet das laufende Programm
        trap   #1         * zurück zum Gem-Desktop
```

```
***************************************************************************
*                                                                         *
* Hier beginnt nun das eigentliche Programm                               *
*                                                                         *
***************************************************************************

haupt:   jsr start1       * Line-A initialisieren
         jsr leerebuf     * Tastaturbuffer leeren
         jsr clear        * Bildschirmlöschen
         jsr init         * Default-Parameter setzten
         jsr gohaupt      * Hauptmenue einstellen
         jsr menuel1      * Ausführung an Menue-Handler über-
haupend: rts              * geben, zurück zum Desktop

***************************************************************************

init: jsr     cursoff           * Cursor ausschalten
      jsr     clear             * Bildschirm löschen
      move.w  #0,wtrack         * Track null, Seite null
      move.w  #0,wside
      move.w  #0,wdrive         * Drive null, Sector eins
      move.w  #1,wsector        * einstellen
      move.w  #0,d0
      move.w  #6,maxdriv        * maximale Anzahl drives
      move.w  #1,maxside        * maximale Anzahl Sides-1
      move.w  #79,maxtrack      * default maximale Anzahl Tracks
      move.w  #9,maxsect        * default maximale " Sectors
      move.w  #9,asector        * maximale Anzahl Sect/Track
      move.b  #'0',setrack      * auch die Menuestrings mit
      move.b  #'9',setrack+1    * den richtigen Werten versorgen
      move.w  #1500,maxclust    * maximale Anzahl Cluster
      move.l  #platztr,editptr  * Buffer
      jsr     prmessag          * Werbetext ausgeben
      rts                       * und zurück
```

```
*************************************************************************
* gibt einen Werbetext mit Copyright aus                                *
*************************************************************************

prmessag: jsr   leerebuf      * Tastaturpuffer leeren
          move.w #20,spalte   * Cursor positionieren
          move.w #10,zeile
          jsr   loccurs
          move.l #hafrag1,a0  * Message Teil 1
          jsr   printf        * ausgeben
          move.w #20,spalte
          move.w #12,zeile
          jsr   loccurs       * Cursor positionieren
          move.l #hafrag2,a0  * Message Teil 2
          jsr   printf        * ausgeben
          move.w #20,spalte
          move.w #14,zeile
          jsr   loccurs       * Cursor positionieren
          move.l #hafrag3,a0  * Message Teil 3
          jsr   printf        * ausgeben
          jsr   wtast         * auf einen Tastendruck warten
          jsr   clear         * Bildschirm löschen
          jsr   leerebuf      * Tastaturbuffer leeren
          rts                 * und zurück

*************************************************************************
* Dies ist die Menue-Loopschleife, der Programmteil, der das            *
* gesammte Programm steuert. Hier wird erkannt, ob durch Cursor         *
* links und rechts ein anderer Menuepunkt angewählt wurde, oder         *
* ob durch Cursor up und down ein Menuepunkt ausgewählt wurde,          *
* dann wird mittels meselct die Programmcontrolle an diesen             *
* Menuepunkt übergeben                                                  *
*************************************************************************

menuel1: jsr   taste          * Tastatur abfragen
         tst.l d0             * keine Eingabe, dann weiter warten
         beq   menuel1
         swap  d0             * sonst auf verschiedene Tasten prüfen
         cmp.b #$44,d0        * F-10 Taste = Ende, NOTSTOP, für Debug
         beq   menende
```

```
menuel2: cmp.b #$4b,d0       * Cursor links
         bne   menuel3
         jsr   curlinks
         bra   menuel1

menuel3: cmp.b #$4d,d0       * Cursor rechts
         bne   menuel4
         jsr   currecht
         bra   menuel1

menuel4: cmp.b #$50,d0       * Cursor down
         bne   menuel5
         jsr   cursdown
         bra   menuel1

menuel5: cmp.b #$48,d0       * Cursur-Up
         bne   menuel6
         jsr   cursup
menuel6: bra   menuel1

haupend2:add.l #8,a7         * Hier sind noch zwei Rücksprung-
menende: rts                 * adressen auf dem Stack (entfernen)


*************************************************************************
* Mit Cursor up wurde eine Menuepunkt ausgewählt, der entsprechende     *
* Sprungadressenblock wird nach jmptable geladen, und zu meselect       *
* verzweigt                                                             *
*************************************************************************


cursup: move.l incvar,jmptable   * Sprungtable für Auswahl durch
        jsr    meselect          * Cursor up, Routine ausführen
        rts                      * und zurück zum Menue-Loop

*************************************************************************
* Es wurde ein Menuepunkt mit Cursor down ausgewählt, in einigen        *
* Menues unterscheiden sich die auszuführenden Unterprogramme je        *
```

```
* nach auswählender Taste (up oder down) z.b inctrack und dectrack  *
* beim Sektormenue                                                  *
*********************************************************************


cursdown: move.l decvar,jmptable       * Sprungtabelle für Auswahl
          jsr    meselect              * druch Cursor down
          rts


*********************************************************************
* Durch Cursorlinks werden die einzelnen Menuepunkte "angefahren" und *
* invers dargestellt                                                *
*********************************************************************

curlinks: move.l revnum,d0             * Anwahl der Menuepunkte
          sub.l  #1,d0
          beq    laround               * durch reverses Schreiben
          move.l d0,revnum             * derselben
          bra    curlend
laround:  move.l ganz,revnum           * swap arround
curlend:  jsr    dispmen               * Anzeige des Menues
          rts

*********************************************************************
* wie bei curlinks, nur wurde Cursor rechts betätigt                *
*********************************************************************

currecht: move.l revnum,d0             * wie bei Curlinks
          add.l  #1,d0
          cmp.l  ganz,d0
          bgt    raround
          move.l d0,revnum
          bra    currend
raround:  move.l #1,revnum
currend:  jsr    dispmen
          rts
```

```
***************************************************************************
* Bringt die entsprechende Unterroutine zur Ausführung                    *
***************************************************************************

meselect: jsr    leerebuf             * Aufruf der ausgewählten
          move.l jmptable,a0          * Routine, in Jumptable ist
          move.l revnum,d0            * die Anfangsadresse des
          subq.l #1,d0                * Sprungadressenblockes
          lsl.l  #2,d0                * mal vier, eine Adresse
          move.l (a0,d0.l),a1         * belegt vier Byte, Laden
          jmp    (a1)                 * und Ausführen der Routine

***************************************************************************
* Handelt die aufgetretenden Fehler, indem aus der auf dem Stack          *
* übergebenen negativen Fehlernummer, ein Fehlerstring gewonnen           *
* wird, der dann angezeigt wird.                                          *
***************************************************************************

errhand: move.w #10,spalte            * Auf dem Stack wird die Fehler-
         move.w #2,zeile              * nummer übergeben (Wort)
         jsr    loccurs               * Cursor in Zeile 2 positionieren
         jsr    delline               * Zeile löschen
         move.w 4(a7),d0              * Fehlernummer holen
         neg.w  d0                    * positiv machen
         cmp.w  #29,d0                * maximal Fehler vergleichen
         blt    errhand1
         move.w #29,d0                * default Fehlernummer

errhand1:lsl.w  #2,d0                 * als Zeiger in die
         move.l #errtab,a1            * Fehlertabelle benutzen
         move.l 0(a1,d0.w),a0         * Fehlerstring holen
         jsr    printf                * drucken
         jsr    wtast                 * auf Tastendruck warten
         jsr    delline               * Zeile wieder löschen
         jsr    cursbuf               * Cursor wieder auf Zeile 4
         move.l (a7)+,a0              * Rücksprungadresse holen
         addq.l #2,a7                 * Stack korrigieren (Fehlern.)
         jmp    (a0)                  * zurück zum Rufer
```

```
**************************************************************************
* Übergibt die Parameter für das Hauptmenue an diverse Variablen        *
* menueadr, incvar, decvar, ganz, revnum                                 *
**************************************************************************

gohaupt: jsr   clear              * Bildschirmlöschen
         move.l #7,ganz           * sieben Menuepunkte im Hauptm.
         move.l #1,revnum         * erster Menuepunkt revers
         move.l #menhaupt,menueadr * Adressen der Menue-Strings
         move.l #haincjmp,incvar  * Adressen der Menuroutinen
         move.l #haincjmp,decvar  * für Cursor up und down
         jsr   dispmen            * gleich, Menue anzeigen
         rts                      * und zurück

**************************************************************************
* Hier folgen die Routinen des Hauptmenues                               *
**************************************************************************

**************************************************************************
* versorgt die Variablen des Menueselect-Systems mit den Adressen       *
* für den TRACK Menuepunkt                                               *
**************************************************************************

gotrack: jsr   clear              * Bildschirm löschen
         move.l #mentrack,menueadr * Adressen der Menue-Strings
         move.l #8,ganz           * das Track-Menue hat 8 Punkte
         move.l #5,revnum         * 5. Punkt revers darstellen
         move.l #trincjmp,incvar  * Cursorup-Jumptable
         move.l #trdecjmp,decvar  * Cursor-down-Jumptable
         jsr   dispmen            * Menue anzeigen
         jsr   cursmess           * und eine Meldung machen
         move.l #trfrag1,a0       * "TRACK MODE"
         jsr   printf
         rts                      * und zurück zum Menue-Handler

**************************************************************************
* versorgt die Variablen des Menueselect-Systems mit den Adressen       *
* für den TRACK with SYNCS Menuepunktes                                  *
**************************************************************************
```

```
gosync:  move.l #6,ganz              * Track mit Sync-Menue hat
         move.l #4,revnum            * sechs Menuepunkte
         move.l #syincjmp,incvar     * up-Jumptable
         move.l #sydecjmp,decvar     * down-Jumptable
         move.l #mensync,menueadr    * Adressen der Menue-Strings
         jsr    dispmen              * anzeigen des Menues
         jsr    cursmess             * Cursor positionieren
         move.l #trfrag2,a0          * "Track with Syncs"
         jsr    printf               * drucken
         rts

***************************************************************************
* versorgt die Variablen des Menueselect-Systems mit den Adressen         *
* für den SECTOR Menuepunkt                                               *
***************************************************************************

gosektor:jsr    clear
         move.l #mensect,menueadr    * Sektormenuepunkte
         move.l #seincjmp,incvar
         move.l #sedecjmp,decvar
         move.l #platztr,editptr
         move.l #8,ganz              * 8 Menuepunkte
         move.l #5,revnum            * 5. revers darstellen
         jsr    dispmen
         jsr    cursmess
         move.l #sefrag1,a0
         jsr    printf
         rts

***************************************************************************
* versorgt die Variablen des Menueselect-Systems mit den Adressen         *
* für den CLUSTER Menuepunkt                                              *
***************************************************************************

goclust: jsr    initdriv             * Clustermenue, erst Drive
         jsr    rdfat                * initialisieren, dann FAT lesen
         move.l #8,ganz              * Menue hat acht Unterpunkte
         move.l #3,revnum            * read = revers
         move.l #menclust,menueadr   * Adresse der Menue-Strings
```

```
        move.l #clincjmp,incvar    * Jumptable
        move.l #cldecjmp,decvar
        jsr    cursmess
        move.l #clfrag1,a0         * "cluster mode"
        jsr    printf              * schreiben
        jsr    dispmen             * Menue anzeigen und
        rts                        * zurück

*************************************************************************
* versorgt die Variablen des Menueselect-Systems mit den Adressen       *
* für den FORMAT Menuepunkt                                             *
*************************************************************************

goformat: jsr clear                 * Format-Menue
        move.l #formmen,menueadr    * Adresse der Menuestrings
        move.l #8,ganz              * acht Menuepunkte
        move.l #3,revnum            * dritten revers
        move.l #foincjmp,incvar     *
        move.l #fodecjmp,decvar
        jsr    dispmen
        jsr    cursmess
        move.l #drfrag1,a0
        jsr    printf
        rts

*************************************************************************
* Untermenue zum Format Menue, versorgt die Variablen mit den           *
* Adressen des GAP's Menue                                              *
*************************************************************************

gogaps:   jsr clear
        move.l #mengap,menueadr
        move.l #7,ganz              * sieben Menuepunkte
        move.l #1,revnum
        move.l #gpincjmp,incvar
        move.l #gpdecjmp,decvar
        jsr    dispmen
        jsr    cursmess
        move.l #gpfrag1,a0
```

```
        jsr    printf
        rts

*************************************************************************
* versorgt die Variablen des Menueselect-Systems mit den Adressen       *
* für den OPTION Menuepunkt                                             *
*************************************************************************

goinit: move.l #6,ganz              * Init Menue hat sechs
        move.l #4,revnum            * Menuepunkte
        move.l #inincjmp,incvar
        move.l #indecjmp,decvar
        move.l #meninit,menueadr
        jsr    dispmen
        jsr    cursmess
        move.l #drifrag1,a0
        jsr    printf
        rts

*************************************************************************
* Hier folgen die ersten Routinen des SECTOR Menuepunktes               *
*************************************************************************

*************************************************************************
* Inkrementiert die Drivezahl innerhalb des Menuepunktes                *
*************************************************************************

incdrive: move.w  wdrive,d0       * aktives drive
          cmp.w   maxdriv,d0      * mit maxdrive vergleichen
          blt     incdr1          * wenn kleiner, dann erhöhen
          move.w  #0,d0           * sonst aktives drive auf null
          bra     incdr2
incdr1:   addq.w  #1,d0
incdr2:   move.w  d0,wdrive       * wieder speichern
          add.b   #'0',d0         * und auch ins Menue eintragen
          move.b  d0,mdrive       *
          jsr     dispmen         * diese auch anzeigen
          rts                     * und zurück
```

```
*****************************************************************************
* Dekrementiert die Drivezahl innerhalb des Menuepunktes, die              *
* folgenden Unterprogramme funktionieren ähnlich, inctrack, incside        *
*****************************************************************************

decdrive: move.w wdrive,d0          * aktuelles drive decrementieren
          cmp.w  #0,d0
          ble    decdr1
          subq.w #1,d0
          bra    decdr2
decdr1:   move.w maxdriv,d0
decdr2:   move.w d0,wdrive
          add.b  #'0',d0
          move.b d0,mdrive
          jsr    dispmen
          rts

*****************************************************************************

incside: move.w  wside,d0           * aktuelle Seite
         cmp.w   #1,d0              * gleich eins?
         blt     incsi1             * wenn ja, dann
         move.w  #0,d0              * Seite null einstellen
         bra     incsi2
incsi1:  move.w  #1,d0              * sonst Seite eins
incsi2:  move.w  d0,wside           * und speichern
         add.b   #'0',d0            * und in Menue-String eintragen
         move.b  d0,mside
         jsr     dispmen            * Menue anzeigen
         rts                        * und zurück

decside: move.w  wside,d0           * Seite decrementieren s.o
         cmp.w   #0,d0
         ble     decsi1
         move.w  #0,d0
         bra     decsi2
decsi1:  move.w  #1,d0
decsi2:  move.w  d0,wside
         add.b   #'0',d0
```

```
          move.b d0,mside
          jsr    dispmen
          rts

*************************************************************************

inctrack: move.w wtrack,d0          * Track inkrementieren, aktueller
          cmp.w  maxtrack,d0        * mit maxtrack vergleichen
          blt    inctr1             * wenn kleiner, dann weiter
          move.w #0,d0              * sonst aktuellen Track auf Null
          bra    inctr2
inctr1:   addq.w #1,d0              * eins addieren
inctr2:   move.w d0,wtrack          * und speichern
          ext.l  d0
          divu   #10,d0             * ins Menue eintragen,
          add.b  #'0',d0            * binär -> ascii
          move.b d0,mtrack          * High-Byte
          swap   d0
          add.b  #'0',d0
          move.b d0,mtrack+1        * Low-Byte
          jsr    dispmen            * Menue anzeigen
          rts

dectrack: move.w wtrack,d0          * Track decrementieren
          cmp.w  #0,d0              * aktueller Track gleich Null,
          ble    dectr1
          subq.w #1,d0
          bra    dectr2
dectr1:   move.w maxtrack,d0        * dann aktueller Track = maxtrack
dectr2:   move.w d0,wtrack
          ext.l  d0
          divu   #10,d0
          add.b  #'0',d0
          move.b d0,mtrack
          swap   d0
          add.b  #'0',d0
          move.b d0,mtrack+1        * in Menue-String eintragen
          jsr    dispmen            * anzeigen und zurück
          rts
```

```
************************************************************************

incsect:  move.w wsector,d0        * aktuellen Sektor inkrementiern
          cmp.w  maxsect,d0        * siehe inctrack
          blt    incse1
          move.w #0,d0
          bra    incse2
incse1:   addq.w #1,d0
incse2:   move.w d0,wsector
          ext.l  d0
          divu   #10,d0
          add.b  #'0',d0
          move.b d0,msector
          swap   d0
          add.b  #'0',d0
          move.b d0,msector+1
          jsr    dispmen
          rts

decsect:  move.w wsector,d0        * aktuellen Sektor dekrementieren
          cmp.w  #0,d0
          ble    decse1
          subq.w #1,d0
          bra    decse2
decse1:   move.w maxsect,d0
decse2:   move.w d0,wsector
          ext.l  d0
          divu   #10,d0
          add.b  #'0',d0
          move.b d0,msector
          swap   d0
          add.b  #'0',d0
          move.b d0,msector+1
          jsr    dispmen
          rts

************************************************************************
* liest den aktuellen Sektor, in wsector, wenn drbyte = 1024 dann     *
* werden 1024 Byte gelesen, da das Betriebssystem mit der              *
* der Anzahl der Sektoren, nur die Anzahl der zu lesenden Bytes        *
```

```
* ausrechnet, indem die Anzahl der Sektoren mit 512 multipliziert      *
* wird. Übergibt man 2 Sektoren, werden also 1024 Byte, gelesen,       *
* egal ob diese in einem Sektor zu 1024 Byte, oder in 2 zu 512         *
* oder in 4 zu 256 Byte organisiert sind.                              *
* Siehe ATARI ST INTERN Seite 399                                      *
************************************************************************

readsec:  move.w drbyte,d0          * Anzahl Bytes/Sektor
          move.w #1,d1              * default gleich 1 Sektor
          cmp.w  #1024,d0           * ist drbyte = 1024, dann
          bne    readweit           * einen Sektor mit 1024 Bytes
          move.w #2,d1              * lesen.
readweit: move.w d1,-(a7)           * Anzahl der Sektoren
          move.w wside,-(a7)        * Seite
          move.w wtrack,-(a7)       * Track
          move.w wsector,-(a7)      * Sektor, bzw. Startsektor
          move.w wdrive,-(a7)       * Drive
          clr.l  -(a7)              * dummy Langwort
          move.l #platztr,-(a7)     * Pufferadresse
          move.w #8,-(a7)           * floprd
          trap   #14                * XBIOS-Call
          add.l  #20,a7             * Stack restaurieren
          tst.w  d0                 * ist ein Fehler aufgetreten?
          bmi    readser            * wenn ja, dann melden
          jsr    showsec            * sonst, gelesenen Sektor anzeigen
          rts                       * und zurück

readser:  move.w d0,-(a7)           * Fehlernummer auf Stack
          jsr    errhand            * Fehler handeln
          jsr    cursmess           *
          move.l #sefrag1,a0
          jsr    printf
          rts                       * und zurück

************************************************************************
* Zeigt den Sektor auf dem Bildschirm, bedient sich der showit         *
* Unterroutine, die alles anzeigt, was ihr übergeben wird, siehe       *
* auch bei editsec                                                     *
************************************************************************
```

```
showsec: move.w #0,head2         * Zeiger in Sektor
        move.l editptr,topptr    * Zeiger auf Sektoranfang
        move.w #31,prcount       * Zähler zum Ausdrucken
        move.w #18,zeicount      * Anzahl der gezeigten Zeilen
        move.w #0,maxdown        * Scrolldown-Flag
        move.w #208,maxup        * Scrollup-Flag
        move.w drbyte,d0         * Bytes in Sektor/ aus Gap-Menue
        cmp.w #1024,d0           * wenn es 1024 sind, dann
        bne   showse2
        move.w #512,maxdown      * Scrollup und Scrolldown-Flags
        move.w #720,maxup        * entsprechend setzten
        move.w #63,prcount       * auch den Druckzeilenzähler
showse2: jsr   showit
        rts

***************************************************************************
* universelle Anzeigecontrol-Routine, die die Tastaturabfrage über-       *
* nimmt, up und down scrollt und auf die 'p' - Taste zweck Drucker-       *
* ausgabe prüft. Es wird ein Zeiger auf den Start des anzuzeigenden       *
* Speicherbereiches übergeben, sowie die obere und untere Begrenzung.     *
***************************************************************************

showit: jsr    cursbuf
        jsr    leerebuf          * Tastaturpuffer leeren
        move.w #0,head2          * Zeiger in Sektor
showit3: move.w head2,head1      * nach Zeiger für dispbuf-Routine
        jsr    dispbuf           * schreibe diesen Buffer
        jsr    leerebuf          * Tastaturbuffer leeren
        jsr    cursbuf
showit4: jsr   taste             * Tastaturabfrage
        swap   d0
        cmp.b  #$19,d0           * Test ob 'p'-Taste betätigt
        beq    printit           * wenn ja, Ausgabe auf Drucker
        cmp.b  #$48,d0           * Test ob Cursor-up
        beq    upper             * wenn ja, handle it
        cmp.b  #$50,d0           * Test ob Cursor-down
        beq    lower             * wenn ja, handle it
        cmp.b  #$1c,d0           * Test ob 'RETURN'-Taste
        beq    shsecli
        cmp.b  #$4b,d0           * Test ob Cursor-links
```

```
         beq    shsecli              * wenn ja
         cmp.b  #$4d,d0              * Test ob Cursor-rechts
         bne    showit4              * wenn nein, zur Frageschleife zur.
         jsr    currecht             * sonst Cursor-rechts, anschl. zur.
         bra    showiten             * zum aufrufenden Programmteil
shsecli: jsr    curlinks             * Cursor-links aufrufen und zurück
showiten:rts                         * zum aufrufenden Programmteil

upper:   move.w head2,d0             * 'handelt' Cursor-up Betätigung
         cmp.w  #0,d0                * zeigt der Pointer auf den Sektor-
         beq    uppend               * Anfang, dann mache nichts
         cmp.w  maxup,d0             * zeigt er auf den oberen Maximal-
         beq    upper1               * punkt, dann subtrahiere 208 zum
                                     * Ausgleich
         sub.w  #256,head2           * sonst subtrahiere 256
         sub.l  #256,topptr          * von Zeiger in Sektor, und Zähler
         bra    uppend               * und zurück
upper1:  sub.w  #208,head2           * subtrahiere 208 von Zeiger in
         sub.l  #208,topptr          * in Sektor zum Angleich an Se.
uppend:  bra    showit3              * und zurück zur Anzeige

lower:   move.w head2,d0             * handling des Cursor-down
         cmp.w  maxup,d0             * wie upper, nur addieren zu
         beq    lowend               * Zeiger und Zähler
         cmp.w  maxdown,d0
         bne    lower1
         add.w  #208,head2
         add.l  #208,topptr
         bra    lowend
lower1:  add.w  #256,head2
         add.l  #256,topptr
lowend:  bra    showit3               * weiter anschauen

*************************************************************************
* Druck den Inhalt des Buffers auf den topptr zeigt als 16 2-stellige  *
* sedezimal Zahlen, und 16 ASCII-Ziffern auf Drucker, die Anzahl        *
* der zu druckenden Zeilen wird in prcount übergeben                    *
*************************************************************************
```

```
printit: move.w  #0,device          * conout auf Drucker
         movem.l a3-a5/d3-d7,savereg * retten der Register
         move.l  #m1secta,a5        * Drucken von aktuellem
         move.w  #45,d7             * Track, Sektor und Seite
printit0:move.b  (a5)+,d0
         move.w  d0,-(a7)
         jsr     conout             * drucken
         dbra    d7,printit0
         move.l  #m1clusa,a5        * Clusternummer drucken
         move.w  #13,d7
printit1:move.b  (a5)+,d0
         move.w  d0,-(a7)
         jsr     conout
         dbra    d7,printit1
         jsr     crlinef            * Carriage-Ret. + Line-Feed
         jsr     crlinef            * 2 mal
         move.l  topptr,a4          * Zeiger in Sektor
         move.l  a4,a5              * speichern
         move.w  head2,head1        * aktuellen Zähler
         move.w  #15,d3             * Spaltenzähler entspricht 16 Sp.
         move.w  d3,d4              * speichern
         move.w  prcount,d5         * Anzahl der zu druckenden Zeilen
printit2:move.w  d4,d3              * Anzahl Zeilen
         jsr     header             * Ausgabe der Zählbytes
         jsr     hex16              * Ausgabe von 16 sedez. Bytes
         move.w  d4,d3              * Zähler restaurieren
         move.l  a5,a4              * Zeiger wieder auf Sektoranfang
         move.w  #5,d7              * fünf Leerzeichen zwischenschieben
printit3:move.w  #$20,-(a7)
         jsr     conout             * Ausgabe
         dbra    d7,printit3
         jsr     char16             * Augabe von 16 ASCII-Zeichen
         add.l   #16,a5             * Zeiger auf Sektor angleichen
         add.w   #16,head1
         jsr     crlinef            * neue Zeile anfangen
         dbra    d5,printit2        * bis alle gewünschten Zeilen
         jsr     leerebuf           * gedruckt wurden
         movem.l savereg,a3-a5/d3-d7 * Rückruf der Register
         move.w  #2,device          * Ausgabe wieder auf Bildschirm
         bra     showit4            * und Anzeige des Sektors
```

```
printerr:rts

****************************************************************************
* schaltet im Sector Mode in den Edit Mode, indem die sehr flexible        *
* editit-Routine aufgerufen wird, dieser Routine wird nur ein              *
* Zeiger auf den Start des zu editierenden Speicherbereichs über-          *
* geben, und zwei Begrenzungsvariablen                                     *
****************************************************************************

editsec: jsr     cursmess
         move.l #edfrag1,a0       * edit message anzeigen
         jsr    printf
         move.w #0,spalte
         move.w #4,zeile
         jsr    loccurs
         jsr    clrest            * restlichen Bildschirm löschen
         move.w drbyte,d0
         cmp.w  #1024,d0          * wenn 1024 Bytes/Sektor dann
         bne    edsewei1
         move.w #512,maxdown      * Begrenzung höher wählen, damit
         move.w #720,maxup        * diese 1024 Bytes auch editiert
         bra    edsewei2          * werden können

edsewei1: move.w #0,maxdown       * sonst entsprechend kleinere
         move.w #208,maxup        * Begrenzungen wählen
edsewei2: move.w #18,zeicount     * 19 Zeilen anzeigen
         move.l #platztr,editptr  * Pufferadresse
         jsr    editit            * und editiern
         jsr    curlinks          * anschließend Sektormenue
         jsr    curlinks          * wieder auf lesen einstellen
         move.w #2,zeile
         jsr    loccurs
         jsr    delline           * Zeile löschen
         jsr    cursmess
         move.l #sefrag1,a0       * Message anzeigen
         jsr    printf
         jsr    cursoff           * Cursor ausschalten
         jsr    showsec           * und den Sektor noch mal anzeigen
```

```
        jsr     leerebuf            * Tastaturpuffer leeren
        rts                         * und zurück

************************************************************************
* Dies ist nun die flexible Editroutine, die beliebig viele Zeilen     *
* a 16 Byte editiert, und diese Zeilen auch als 16 sedezimal und       *
* 16 ASCII Zeichen anzeigt                                             *
************************************************************************

editit: movem.l a3-a6/d3-d7,-(a7)  * Register retten
        move.l  editptr,topptr     * Pufferadresse
        move.w  #0,head2           * Zähler für Puffer initialisieren
        move.w  #0,head1
        jsr     dispbuf            * erste Pufferseite anzeigen
        jsr     leerebuf           * Tastaturbuffer leeren
        move.w  #7,spalte          * Edit beginnt in Spalte 7
        move.w  #4,zeile
        jsr     loccurs            * Cursor positionieren
edits0: jsr     curson             * blinkenden Cursor einschalten
        move.l  #retw1,-(a7)       * Adresse eine Variablen an
        jsr     hexein             * hexein übergeben, in dieser
        jsr     cursoff            * Adresse steht dann die
        tst.w   retw1              * eingegebene Zahl, war die Zahl
        bmi     otherkey           * negativ, dann andere Taste bet.
        move.w  zeile,d0           * aktuelle Zeile
        subq.w  #4,d0              * Startoffset der ersten Zeile
        lsl.w   #4,d0              * mal 16 Zeichen / Zeile
        move.w  spalte,d2          * plus Spalte - Startoffset
        sub.w   #7,d2
        ext.l   d2                 * dividiert durch 3 Zeichen pro
        divu    #3,d2              * Byte (1 Space + 2 Digits)
        add.w   d2,d0              * plus Zeilenoffset
        move.w  retw1,d1           * eingegebene Sedezimalziffer
        move.l  topptr,a3          * Startadresse des Puffers
        move.b  d1,0(a3,d0.w)      * Ziffer in Puffer eintragen, mit
        jsr     dispzeil           * Offset als Zeiger, Zeile anzeigen
        cmp.w   #52,spalte         * war es das letzte Editbyte in der
        blt     edits1             * Zeile ?
        move.w  #4,spalte          * wenn ja, an Zeilenanfang zurück
```

```
edits1:  addq.w   #3,spalte       * sonst 3 Zeichen/Byte addieren
         jsr      loccurs         * Cursor positionieren
         bra      edits0          * und weiter editieren

***********************************************************************

otherkey: move.l  varl1,d0        * hierhin wird verzweigt, wenn
         swap     d0              * keine gültige Ziffer eingegeben
         cmp.b    #$4b,d0         * wurde,  Cursor left?
         beq      oleft           * ja, handle it
         cmp.b    #$4d,d0         * cursor right
         beq      oright
         cmp.b    #$50,d0         * Cursor down
         beq      odown
         cmp.b    #$48,d0         * cursor up
         beq      oup
         cmp.b    #$52,d0         * Insert-Taste
         beq      edend1
         cmp.b    #$72,d0         * Enter-Taste
         beq      edend1          * beendet edit Mode
         cmp.b    #$1c,d0         * Return-Taste beendet ebenfalls
         beq      edend1          * den Edit-Mode
         jsr      dispzeil        * sonst Zeile anzeigen
         jsr      loccurs         * Cursor positionieren
         bra      edits0          * und weiter editieren
oleft:   move.w   spalte,d0       * Cursor left
         cmp.w    #7,d0           * Cursor schon am linken Rand,
         bgt      oleft1          * wenn nicht, dann weiter
         move.w   #55,spalte      * wenn ja, wrap arround
oleft1:  subq.w   #3,spalte       * wenn nicht, 3 Zeichen/Byte
         jsr      loccurs         * subtrahieren, Cursor positioniern
         jsr      leerebuf        * Tastaturbuffer leeren
         bra      edits0          * und weiter editieren

oright:  move.w   spalte,d0       * das gleiche in grün für Cursor
         cmp.w    #52,d0          * right
         blt      oright1
         move.w   #4,spalte
oright1: addq.w   #3,spalte
         jsr      loccurs
```

```
        jsr     leerebuf
        bra     edits0

*****************************************************************

odown:  jsr     cursoff             * Cursor down betätigt,
        move.w  zeile,d0            * aktuelle Zeile
        cmp.w   #22,d0              * kleiner als 22
        blt     odown2              * wenn ja, weiter
        move.w  head2,d0            * sonst Zähler mit
        cmp.w   maxdown,d0          * Begrenzung vergleichen
        bne     odown1              * wenn nicht gleich weiter
        add.w   #208,head2          * wenn gleich, den Rest des
        add.l   #208,topptr         * Buffers bearbeiten, darum nur
        move.w  spalte,oldspa1      * 208 anstelle von 256 addieren
        jsr     dispbuf             * den Buffer erst anzeigen
        move.w  oldspa1,spalte      * Cursor in alte Spalte
        move.w  #5,zeile            * Offset in Buffer
        jsr     loccurs             * Cursor positionieren
        bra     odownend            * und zurück
odown1: cmp.w   maxup,d0            * wenn gleich oberer Begrenzung
        beq     odownend            * dann mache nichts
        add.w   #256,head2          * sonst addiere 256 zu den
        add.l   #256,topptr         * Zeigern
        move.w  spalte,oldspa1
        jsr     dispbuf             * und zeige den Buffer an
        move.w  oldspa1,spalte
        move.w  #6,zeile
        jsr     loccurs
        bra     odownend            * und zurück
odown2: addq.w  #1,zeile            * wenn nicht in Zeile 22, dann
        jsr     loccurs             * erhöhe aktuelle Zeile um eins
odownend: jsr   leerebuf            * leere Tastaturbuffer und
        bra     edits0              * weiter editieren

*********************************************************************

oup:    jsr     cursoff             * das gleich wie für Cursor-down
        move.w  zeile,d0            * nun für Cursor up
        cmp.w   #4,d0               * aktuelle Zeile = Zeile 4
```

```
         bne      oup2              * wenn nicht, dann weiter
         move.w   head2,d0          * wenn ja, Zähler laden
         cmp.w    #0,d0             * am oberen Ende des Editpuffers?
         beq      oupend            * wenn ja, mache nichts
         cmp.w    maxup,d0          * wenn nein, vergleiche mit Begre.
         beq      oup1              * wenn gleich subtrahiere nur 208
         sub.w    #256,head2        * sonst subtrahiere 256 von den
         sub.l    #256,topptr       * Zeigern,
         move.w   spalte,oldspa1    * alte Spalte laden
         jsr      dispbuf           * Buffer anzeigen
         move.w   oldspa1,spalte    * Cursor in alte Spalte
         move.w   #19,zeile         * Offset in Zeile
         jsr      loccurs           * Cursor positionieren
         bra      oupend            * und zum Ende
oup1:    sub.w    #208,head2        * oberen Rest des Buffers anzeigen
         sub.l    #208,topptr
         move.w   spalte,oldspa1
         jsr      dispbuf
         move.w   oldspa1,spalte
         move.w   #19,zeile
         jsr      loccurs
oup2:    subq.w   #1,zeile          * wenn nicht in oberster Zeile,
         jsr      loccurs           * einfach Zeile decrementieren
oupend:  jsr      leerebuf          * Tastaturbuffer leeren
         bra      edits0            * und weiter editieren
edend1:  move.w   #0,spalte         * Editieren abbrechen, Cursor
         move.w   #4,zeile          * in Zeile 4 positionieren
         jsr      loccurs
         movem.l  (a7)+,a3-a6/d3-d7 * Register zurück
         rts                        * und zurück


*************************************************************************


writsec: movem.l  a3-a6/d3-d7,-(a7) * schreibt einen Sektor auf Disk
         move.w   #0,spalte         * erst einmal fragen ob wirklich
         move.w   #2,zeile          * geschrieben werden soll
         jsr      loccurs
         move.l   #wrfrag1,a0
```

```
        jsr     printf
        move.l  #m1secta,a3     * aktuellen Track,Sektor ect.
        move.w  #45,d3          * auf Bildschirm ausgeben
writl1: move.b  (a3)+,d0
        move.w  d0,-(a7)
        jsr     conout
        dbra    d3,writl1
        move.l  #wrfrag2,a0
        jsr     printf
        jsr     leerebuf        * Tastaturpuffer leeren, und
        jsr     wtast           * auf Tastendruck warten
        cmp.b   #'y',d0         * wurde weder 'y' noch 'Y'
        beq     writit          * betätigt, dann nicht schreiben
        cmp.b   #'Y',d0
        bne     wrend1          * und zum Ende hüpfen

writit: move.w  drbyte,d0       * wenn drbyte = 1024, dann die
        cmp.w   #1024,d0        * selbstgeschriebene writesec-
        beq     selfsect        * Routine benutzten
        move.w  #1,d1           * sonst einen Sektor schreiben
writi1: move.w  d1,-(a7)        * Anzahl
        move.w  wside,-(a7)     * Seite
        move.w  wtrack,-(a7)    * Track
        move.w  wsector,-(a7)   * Sector
        move.w  wdrive,-(a7)    * Drive
        clr.l   -(a7)           * Dummy-Langwort
        move.l  #platztr,-(a7)  * Pufferadresse
        move.w  #9,-(a7)        * flopwr
        trap    #14             * XBIOS-Call
        add.l   #20,a7
        tst.w   d0              * Test ob Fehler
        bmi     writerr         * Fehler handeln

wrend2: jsr     delline
        jsr     leerebuf
        jsr     cursmess        * Mode anzeigen
        move.l  #sefrag1,a0
        jsr     printf
        movem.l (a7)+,a3-a6/d3-d7
        rts                     * und zurück
```

```
wrend1:  jsr      delline
         move.l   #wrfrag3,a0       * message = "not written"
         jsr      printf
         jsr      leerebuf
         jsr      wtast
         bra      wrend2

writerr: move.w   d0,-(a7)          * Fehler handeln
         jsr      errhand
         bra      wrend2


**********************************************************************
**********************************************************************
*   Hier geht es los mit der modularen Assemblerprogrammierung       *
*   geben Sie einfach die Routinennamen mit dem RTS ein. Im          *
*   Programm funktioniert dann nur das Sektormenue, d.h. Sie         *
*   können dann nur Sektoren lesen und schreiben, wenn dieses        *
*   Rumpfprogramm dann fehlerfrei funktioniert, können Sie die       *
*   vollständigen Unterprogramme aus dem jeweiligen Listing          *
*   hier eintragen. Am besten tragen Sie jeweils den ganzen, zu      *
*   einem Menuepunkt gehörenden Block ein, da die meisten            *
*   Menuepunkte auf Routinen der anderen Punkte zurückgreifen        *
*   ein wenig sollten Sie auch auf die Reihenfolge der Implentation  *
*   achten, als Empfehlung schlage ich vor: erste OPTION, dann          *
*   TRACK, anschließend TRACK with SYNCS, FORMAT und CLUSTER         *
**********************************************************************
**********************************************************************


**********************************************************************
**********************************************************************
*  Subroutines des Menuepunktes OPTION, sollten zuerst implementiert *
*  werden, da hiermit auf die Möglichkeit gegeben wird den 10. Sektor *
*  auf dem 82 Track ect. zu lesen, außerdem werden einige Routinen   *
*  von anderen Programmteilen aufgerufen                             *
**********************************************************************
**********************************************************************
```

```
incmaxtr: rts

decmaxtr:rts

incmaxse:rts

decmaxse:  rts

dodrivin: rts

showbpb: rts

initdriv: rts

rdfat:    rts

**********************************************************************
**********************************************************************
*  Subroutines des Menuepunktes TRACK des Hauptmenues plus           *
*  einer eigenen Sektorschreibroutine                                *
**********************************************************************
**********************************************************************

**********************************************************************
*  eigene Sektorschreibroutine, greift direkt auf Controller und DMA *
*  chip zu. die XBIOS-Routine zum Sektorschreiben läßt sich im Gegen- *
*  satz zur Sektorleseroutine nicht dazu "überreden" Sektoren mit    *
*  1024 Byte zu schreiben, daher wird diese Routine aufgerufen. In   *
*  der Grundaustattung des Programms (nur mit Sektormenue) ist es    *
*  leider nicht möglich Sektoren mit 1024 Bytes pro Sektor zu        *
*  schreiben.                                                        *
**********************************************************************

selfsect: rts
```

```
****************************************************************************
*  Subroutines des Menuepunktes TRACK des Hauptmenues                      *
****************************************************************************


read1tr: rts

incstra: rts

decstra: rts

edittr:  rts

showtr:  rts

writ1tr: rts

*************************************************************************
*************************************************************************
*  Subroutines des Menuepunktes TRACK with SYNCS, die Routinen         *
*  greifen auf keine anderen Routinen zurück, kann daher wahlfrei      *
*  implementiert werden                                                 *
*************************************************************************
*************************************************************************

rdtracks: rts

shtracks: rts

readadr:  rts

showadr:  rts

****************************************************************************
*  Subroutines des Menuepunktes CLUSTER des Hauptmenues, die Routinen *
*  greifen auf Routinen des OPTION-Menue-Punktes zurück, daher bitte  *
*  das Option-Menue zuerst implementieren                             *
****************************************************************************
```

```
edclust:  rts

decclust: rts

incclust: rts

nextclst: rts

wrclust: rts

rdclust: rts

stclust: rts

***************************************************************************
*   Format Unterroutinen des Hauptmenues                                  *
*   diese Routinen greifen auf Unterprogramme des Menuepunktes            *
*   TRACK with SYNCS zurück, daher muß der Menuepunkt TRACK with          *
*   SYNCS zuerst implementiert werden, und danach erst der Formatter      *
***************************************************************************

format1: rts

xformat: rts

incgap1: rts

incgap2: rts

incgap3: rts

incgap4: rts

incgap5: rts

decgap1: rts

decgap2:  rts
```

```
decgap3: rts

decgap4:rts

decgap5:rts


incbyte:rts

decbyte: rts

**************************************************************************
**************************************************************************
*                                                                        *
*  Hier folgen einige sehr oft benötigte Unterprogramme                  *
*                                                                        *
**************************************************************************
**************************************************************************

**************************************************************************
*  zeichnet eine Horizontale Linie von 0,20 nach 639,20                  *
*~************************************************************************

hline:  move.l   lineavar,a0        * Zeiger auf Line-A-Variablen
        move.w   #0,38(a0)          * X1
        move.w   #20,40(a0)         * Y1
        move.w   #639,42(a0)
        move.w   #1,24(a0)          * Farbe
        move.w   #0,36(a0)          * Write-Mode
        move.l   #pattern,46(a0)    * Muster + Anzahl Musterworte
        move.w   #0,50(a0)          * Horizontal Line
        dc.w     $a004
        rts


**************************************************************************
*  Schreibt einen String auf den Bildschirm                              *
**************************************************************************
```

```
printf:  move.l   a0,-(a7)           * schreibt den String, dessen
         move.w   #9,-(a7)           * Anfangsadresse sich imAdress-
         trap     #1                 * register A0 befindet, auf den
         addq.l   #6,a7              * Bildschirm, String muß mit Null
         rts                         * abgeschlossen werden

**********************************************************************
* Initialisiert die Line-A-Variablen, und speichert die Adresse des  *
* Variablenblocks in "lineavar".                                     *
**********************************************************************

inlinea: dc.w    $a000          * initialisiert die Line-A-Variabl.
         move.l   a0,lineavar   * Adresse speichern
         move.w   #0,32(a0)
         move.w   #$ffff,34(a0) * Muster der Linie
         move.w   #0,36(a0)      * Writing mode = replace
         move.w   #1,24(a0)      * Zeichenfarbe
         rts


********************************************************************
*  Line-A- initialisieren                                          *
********************************************************************

start1:  jsr      inlinea            * Line A initialisieren
         rts


**********************************************************************
*                                                                    *
*  Schreibt eine Zahl die in Wortgröße auf dem Stack übergeben wird *
*  als 2-stellige Sedezimalzahl auf den Bildschirm, bzw. auf das     *
*  durch den Inhalt von device bestimmte Gerät                       *
*                                                                    *
**********************************************************************


hexpr:   move.w   4(a7),d1      * Argument vom Stack holen
         and.w    #$00ff,d1     * Highwort ausmaskieren
         move.w   d1,varw1       * Byte speichern
```

```
         lsr.w    #4,d1           * unteres Nibbel "rausschieben"
         ext.w    d1              * auf Wortgröße erweitern
         and.w    #$00ff,d1       * high-Byte ausmaskieren
         cmp.w    #9,d1           * größer als neun, dann
         bgt      ischar1         * einen Character von 'A'-'F' drucken
         jsr      hexdig          * sonst eine Ziffer von '0'-'9'
         bra      secdig1
ischar1: jsr      hexchar
secdig1: move.w   varw1,d1        * nun das untere Nibble des low-Byte
         and.w    #$000f,d1       * umwandeln, ausmaskieren der oberen
         cmp.w    #9,d1           * größer als neu, siehe oben
         bgt      ischar2
         jsr      hexdig
         bra      hexpren         * zum ende

ischar2: jsr      hexchar
hexpren: move.l   (a7)+,a0        * Rücksprungadresse noch auf dem STack
         add.l    #2,a7           * Stack vom Variablen befreien
         jmp      (a0)            * und zurück zum Rufer, wie rts


hexdig:  add.w    #48,d1          * ASCII-Wert von '0' addieren
         move.w   d1,-(a7)        *
         move.w   device,-(a7)    * und drucken
         move.w   #3,-(a7)        * conout
         trap     #13             * BIOS-TRAP
         addq.l   #6,a7
         rts

hexchar: sub.w    #10,d1          * Zehn abziehen und ASCII-Wert von
         add.w    #65,d1          * 'A' addieren
         move.w   d1,-(a7)
         move.w   device,-(a7)    * und drucken
         move.w   #3,-(a7)
         trap     #13
         addq.l   #6,a7
         rts
```

```
**************************************************************************
*  Ausgabe einer 2 Byte Integerzahl als Dezimalzahl                      *
**************************************************************************


dezpr:  move.w  #0,dflag       * drucken eine 2-Byte Integer Dezimal-
        move.w  4(a7),d3       * zahl, führende Nullen werden unter-
        ext.l   d3             * drückt und als Spaces ausgegeben
        divs    #10000,d3
        beq     dezpr1         *
        move.w  #-1,dflag      * Flag für geforderte Ausgabe, keine
dezpr1: jsr     deznum         * drucken
        swap    d3             * Rest der 1. Division nun durch 1000
        ext.l   d3             * dividieren
        divs    #1000,d3       * und dann als 1000'er Stelle drucken
        beq     dezpr3
dezpr2: move.w  #-1,dflag      * ungleich Null, dann Flag setzten
dezpr3: jsr     deznum
        swap    d3
        ext.l   d3             * Rest durch 100 dividieren
        divs    #100,d3
        beq     dezpr4
        move.w  #-1,dflag
dezpr4: jsr     deznum         * und drucken
dezpr5: swap    d3
        ext.l   d3
        divs    #10,d3         * Rest durch 10 dividieren und
        beq     dezpr7
        move.w  #-1,dflag
dezpr7: jsr     deznum         * als Zehnerstelle der Zahl drucken
        swap    d3             * den Rest der letzten Division auf
        move.w  #-1,dflag      * jeden Fall drucken, da bei Null-
        jsr     deznum         * Ergebnis auch eine Null angezeigt
        move.l  (a7)+,a0       * werden soll.
        addq.l  #2,a7          * Rücksprungadresse in A0 holen, STack
        jmp     (a0)           * restaurieren, und zurück zum Rufer
```

```
deznum:  tst.w    dflag          * druckt eine Ziffer von '0'-'9'
         bne      deznum1        * aber nur, wenn dflag ungleich null
         move.b   #' ',d0        * ist, durch
         move.w   d0,-(a7)
         bra      deznum2
deznum1: add.b    #'0',d3        * Addition von ASCCI-Wert für '0'
         move.w   d3,-(a7)
deznum2:  jsr      conout        * drucken
         rts


***********************************************************************


dezlpr:   move.w   #0,dflag      * druckt eine 4-Byte Integerzahl, die
         move.l   4(a7),d3       * auf dem Stack als Langwort über-
         move.l   d3,d4          * geben wird, als Dezimalzahl,
         divs     #10000,d3      * führende Nullen werden ebenfalls
         ext.l    d3             * nicht berücksichtigt, und als
         divs     #10,d3         * spaces gedruckt.
         move.w   d3,d5          * erst durch 100000 dividieren.
         tst.w    d3       * wenn null, dann keine 100000'er Stelle
         beq      dezlpr1
         move.w   #-1,dflag
dezlpr1:  jsr      deznum        * sonst die 100000'er St. drucken
         move.w   d5,d3          * Ergebnis der Division wieder mit
         muls     #10,d3         * 100000 multiplizieren, Rest zur
         muls     #10000,d3      * Ausgangszahl wie bei dezpr weiter
         sub.l    d3,d4          * behandeln.
         move.l   d4,d3
         divs     #10000,d3
         beq      dezlpr3
dezlpr2: move.w   #-1,dflag
dezlpr3: jsr      deznum
         swap     d3
         ext.l    d3
         divs     #1000,d3
         beq      dezlpr4
         move.w   #-1,dflag
```

```
dezlpr4: jsr      deznum
         swap     d3
         ext.l    d3
         divs     #100,d3
         beq      dezlpr5
         move.w   #-1,dflag
dezlpr5: jsr      deznum
         swap     d3
         ext.l    d3
         divs     #10,d3
         beq      dezlpr6
         move.w   #-1,dflag
dezlpr6: jsr      deznum
         swap     d3
         move.w   #-1,dflag
         jsr      deznum
         move.l   (a7)+,a0
         addq.l   #4,a7
         jmp      (a0)

**********************************************************************
*   Anzeige der Menueleiste                                          *
**********************************************************************

dispmen: move.w #0,spalte        * Cursor auf oberste Zeile
         move.w #0,zeile
         jsr    loccurs          * positionieren
         move.l  menueadr,a6     * hier steht ein Zeiger auf die
         move.l  revnum,d6       * Adressen der Menuestrings
         subq.l  #1,d6           * in revnum befindet sich die
         beq     dispwei1        * Nummer des revers dargestellten
         subq.l  #1,d6           * Menuepunktes
dispmen1: move.l  (a6)+,a0       * die einzelnen Adressen holen, und
         jsr     printf          * mittel printf drucken, bis alle
         dbra    d6,dispmen1  * Menuestrings bis zum Reversen gedrukt
dispwei1: jsr     revon          * dann den reversen Menuestring
         move.l  (a6)+,a0        * anzeigen,
         jsr     printf          * revers wieder ausschalten
         jsr     revout
```

```
        move.l    ganz,d7            * Gesamtzahl aller Menuepunkte
        sub.l     revnum,d7          * minus der Reversen Nummer
        beq       dispmen3           * wenn Null, dann war es letzter
        subq.l    #1,d7              * sonst die restlichen Menue-
dispmen2: move.l   (a6)+,a0          * punkte nichtrevers drucken
        jsr      printf
        dbra      d7,dispmen2        * bis alle gedruckt sind
dispmen3: jsr      hline             * zu guter letzt, eine horizontale
        jsr      delrest             * Linie zeichnen, und zurück
        rts


*************************************************************************
*                                                                       *
*   Schreibt den Inhalt eines Speicherbereiches, dessen Anfangs-        *
*   adresse in topptr übergeben wird, als 16 2-stellige sedez. Zahl    *
*   sowie 16 zugehörige ASCCI-Zeichen auf den Bildschirm                *
*************************************************************************


dispbuf: movem.l  a3-a5/d3-d7,savereg  * Register speichern
         move.l   topptr,a4            * Startadresse des Speicherber.
         move.l   a4,a5                * zwischenspeichern
         move.w   head2,head1          * Zähler für offset in Block
         move.w   #15,d3               * Spaltenzähler = 16 Spalten
         move.w   d3,d4                * zwischenspeichern
         move.w   zeicount,d5          * Anzahl der Zeilen wird in
         move.w   #4,zeile             * zeicount übergeben
         move.w   #4,curzei1           * Cursor auf Zeile 4 Spalte 0
         move.w   #0,spalte
         jsr      loccurs              * positionieren

dispb1:  move.w   #0,spalte            * Cursor auf Current-zeile
         move.w   curzei1,zeile        * positionieren
         move.w   d4,d3                * Spaltenzähler
         jsr      loccurs
         jsr      header               * Zähler in Block drucken
         jsr      hex16                * 16 Sedezimalzahlen drucken
         move.w   d4,d3                * Spaltenzähler
         move.l   a5,a4                * topptr
```

```
        move.w  #59,spalte          * in Spalte 59 die Ascii-Zeichen
        move.w  curzei1,zeile
        jsr     loccurs
        jsr     char16              * 16 Ascii-Zeichen drucken
        add.l   #16,a5              * 16 zum Zeiger in Speicher add.
        add.w   #1,curzei1          * in der nächsten Zeile weiter
        add.w   #16,head1           * machen, 16 zum Zähler addieren
        dbra    d5,dispb1           * bis alle Zeilen angezeigt
        jsr     leerebuf            * Tastaturpuffer leeren
        movem.l savereg,a3-a5/d3-d7 * und die Register zurückholen
        rts

**************************************************************************
*   schreibt einen Header vor jede Zeile                                 *
**************************************************************************

header: move.w  head1,d6            * Zähler
        lsr.w   #8,d6               * durch 256 dividieren (High-Byte)
        move.w  d6,-(a7)            * und als sedezimal-Zahl drucken
        jsr     hexpr
        move.w  head1,-(a7)         * Low-Byte drucken
        jsr     hexpr
        move.b  #':',d6             * Doppelpunkt drucken
        move.w  d6,-(a7)
        jsr     conout
        rts

**************************************************************************
*   schreibt 16 sedezimal-Zahlen                                         *
**************************************************************************

hex16:  move.w  #$20,-(a7)          * zwei Spaces drucken
        jsr     conout
        move.w  #$20,-(a7)
        jsr     conout
hex161: move.b  (a4)+,d7            * den Inhalt des Speicherbereichs
        move.w  d7,-(a7)            * als Sedezimalzahlen Drucken
        jsr     hexpr               * jeweils 16 mit einem Space dazwi.
        move.w  #$20,-(a7)
```

```
        jsr     conout          * der Zähler wird in d3 übergeben
        dbra    d3,hex161
        rts

*****************************************************************************
*   schreibt 16 ASCII-Zeichen auf den Bildschirm                           *
*****************************************************************************

char16: move.b  #':',d7         * erst einen Doppelpunkt und
        move.w  d7,-(a7)
        jsr     conout
        move.w  #$20,-(a7)      * zwei Spaces. dann
        jsr     conout
char161: move.b (a4)+,d7        * 16 Ascii-Zeichen drucken
        cmp.b   #$20,d7         * alles was kleiner als $20
        bgt     char162         * als Punkt drucken
        move.b  #'.',d7
char162: ext.w  d7              * sonst High-Byte ausmaskieren
        and.w   #$00ff,d7
        move.w  d7,-(a7)
        jsr     conout          * und drucken
        dbra    d3,char161      * 16 mal, in D3 übergeben
        rts


*****************************************************************************
*   schreibt eine ganze Zeile auf den Bildschirm                           *
*****************************************************************************

dispzeil: move.w spalte,oldspa1  * eine Zeile im 16/16 Format
        move.w  #0,spalte       * drucken
        jsr     loccurs         * Cursor positionieren
        move.w  oldspa1,spalte
        move.w  #15,d3          * jeweils 16 Spalten
        move.w  d3,d4
        move.l  topptr,a4       * Zeiger auf Anfang des Speicher-
        clr.l   d0              * bereiches, mit Hilfe der
        move.w  zeile,d0        * momentanen Zeile, die Position
        subq.w  #4,d0           * relativ zum Anfang des Bereiches
        lsl.w   #4,d0           * berechnen:   mal 16
```

```
        move.w  d0,d1              * zwischenspeichern
        add.w   head2,d0           * Zähleroffset addieren
        move.w  d0,head1           * in aktuellen Zähler übernehmen
        ext.l   d1
        add.l   d1,a4              * zum Zeiger in Speicherbereich
        move.l  a4,a5              * addieren, gleich Zeiger auf
        jsr     header             * die momentan editierte Zeile
        jsr     hex16              * 16 Zahlen drucken
        move.w  spalte,oldspa1
        move.w  #59,spalte
        jsr     loccurs
        move.w  d4,d3
        move.l  a5,a4              * und 16 ASCII's drucken
        jsr     char16
        move.w  oldspa1,spalte     * alte spalte zurückholen, und
        jsr     loccurs            * Cursor positionieren
        rts

***********************************************************************
*   Hier folgen die Routinen für die Terminalemulation                *
***********************************************************************


revon:  move.l   #revers1,a0       * Reverses drucken ab hier ein
        jsr      printf
        rts

revout: move.l   #revers2,a0       * revers ab hier wieder aus
        jsr      printf
        rts

delrest: move.l  #clrest2,a0       * löscht den Rest der Zeile
        jsr     printf
        rts

delline: move.l  #delline1,a0      * löscht die ganze Zeile
        jsr     printf
        rts
```

```
clear:    move.l   #clear1,a0        * löscht den ganzen Bildschirm
          jsr      printf            * und positioniert Cursor in
          rts                        * obere linke Ecke

home:     move.l   #home1,a0         * positioniert Cursor in obere
          jsr      printf            * linke Ecke
          rts

crlinef:  move.w   #$1c,-(a7)        * gibt Carriage-Return mit
          jsr      conout            * Linefeed auf Ausgabegerät
          move.w   #$0a,-(a7)
          jsr      conout
          rts


clrest:   move.l   #clrest1,a0       * löscht den Rest des Bildschirms
          jsr      printf
          rts

curson:   move.l   #curon1,a0        * schaltet Cursor ein
          jsr      printf
          rts

cursoff:  move.l   #curout1,a0       * schaltet den Cursor aus
          jsr      printf
          rts

cursmess: move.w   #30,spalte        * positioniert den Cursor
          move.w   #2,zeile          * zur Ausgabe von Mitteilungen
          jsr      loccurs
          rts

cursbuf:  move.w   #0,spalte         * positioniert den Cursor zur
          move.w   #4,zeile          * Ausgabe des Sektorbuffers
          jsr      loccurs
          rts
```

```
**************************************************************************
*     Cursorpositionierung                                               *
**************************************************************************

loccurs: move.l   #loccurs1,a0      * positioniert den Cursor auf die
         addq.l   #2,a0             * in spalte und zeile übergebenen
         move.w   zeile,d0          * Koordinaten, (0-79),(0-24)
         add.w    #32,d0            * internen Offset addieren
         move.b   d0,(a0)+          * Speichern
         move.w   spalte,d0
         add.w    #32,d0            * internen Offset addieren
         move.b   d0,(a0)+          * und speichern
         move.l   #loccurs1,a0      * anschließend veränderten
         jsr      printf            * Positionierbefehl drucken
         rts

curstab: move.w   tab1,spalte       * positioniert den Cursor in
         jsr      loccurs           * Spalte tab1 der aktuel. Zeile
         rts

**************************************************************************
*    Tastaturabfrage, wartet nicht und gibt sowohl Tastencode als       *
*    auch ASCII-Code ind D0 zurück, wurde keine Taste betätigt,         *
*    ist D0 gleich null                                                 *
**************************************************************************

taste:   move.w   #2,-(a7)          * fragt die Tastatur ab
         move.w   #1,-(a7)          * gibt den ASCII-Code der
         trap     #13               * betätigten im Low-Byte des
         addq.l   #4,a7             * unteren Wortes von D0 und den
         tst.w    d0                * Scan-Code im Low-Byte des oberen
         bpl      endtast2          * Wortes von D0 zurück.
         move.w   #2,-(a7)          * wurde Taste betätigt, dann
         move.w   #2,-(a7)          * Taste aus Buffer holen und
         trap     #13               * zurück
         addq.l   #4,a7
         rts
```

```
endtast2: move.l    #0,d0           * sonst Null zurück
          rts

leerebuf: move.w    #$b,-(a7)       * leert den Tastaturbuffer
          trap      #1
          addq.l    #2,a7
          tst.w     d0
          beq       leeren1
          move.w    #7,-(a7)
          trap      #1
          addq.l    #2,a7           * solange wiederholen, bis kein
          bra       leerebuf        * Zeichen mehr im Puffer
leeren1:  rts

conout:   move.w    4(a7),d0        * gibt ein Zeichen auf das in
          move.w    d0,-(a7)        * device stehende Gerät aus
          move.w    device,-(a7)    * siehe ST INTERN
          move.w    #3,-(a7)
          trap      #13
          addq.l    #6,a7
          move.l    (a7)+,a0        * Rücksprungadresse holen
          addq.l    #2,a7
          jmp       (a0)

wtast:    move.w    #1,-(a7)        * Tastatureingebe, warte auf
          trap      #1              * die Eingabe, und zeigt einen
          addq.l    #2,a7           * blinkenden Cursor
          rts

*************************************************************************
*   Eingabe einer 1 Byte Sedezimal-Zahl an die auf dem Stack über      *
*   gebenen Adresse                                                     *
*************************************************************************

hexein:   jsr       wtast           * gibt an die Variable auf dem
          move.l    d0,varl1        * Stack eine über die Tastatur
          cmp.b     #'f',d0         * einzugebende Hexzahl (2 digits)
          bgt       hexeierr        * wurde eine unerlaubte Taste
          cmp.b     #'a',d0         * betätigt, so wird -1 übergeben
          blt       hexein1
```

```
        sub.b   #'a',d0         * Test ob zwischen 'a' und 'f'
        add.b   #10,d0          * wenn ja 'a' subtrahiern, und 10
        bra     hexein2         * addieren.
hexein1: cmp.b  #'0',d0         * wenn nicht, dann Test ob zwichen
        blt     hexeierr        * null und neun,
        cmp.b   #'9',d0
        bgt     bstest1       * wenn nicht, dann Test ob('A'-'F')
        sub.b   #'0',d0         * sonst '0' subtrahieren
hexein2: lsl.w  #4,d0           * mal 16 = High-nibble
        move.w  d0,varw1        * zwichenspeichern
        jsr     wtast           * nächstes Nibble holen
        move.l  d0,varl1        * zwischenspeichern
        cmp.b   #'f',d0         * gleicher Test wie erstes Nibble
        bgt     hexeierr
        cmp.b   #'a',d0
        blt     hexein3
        sub.b   #'a',d0
        add.b   #10,d0
        bra     hexein4
hexein3: cmp.b  #'0',d0
        blt     hexeierr
        cmp.b   #'9',d0
        bgt     bstest2         * Test ob Großbuchstaben
        sub.b   #'0',d0
hexein4: move.w varw1,d1
        or.w    d1,d0
        ext.w   d0
        and.w   #$00ff,d0
        move.w  d0,varw2       * ohne Fehler zurück
hexein5: move.l 4(a7),a0
        move.w  d0,(a0)
        move.l  (a7)+,a0
        addq.l  #4,a7
        jmp     (a0)            * back to caller

hexeierr: move.w #-1,d0
        bra     hexein5         * mit Fehlercode zurück

bstest1: cmp.b  #'F',d0         * Test ob zwischen 'A' und 'F'
        bgt     hexeierr        * wenn nein, dann mit Fehlercode
```

```
        cmp.b   #'A',d0         * zurück
        blt     hexeierr        * sonst ascii 'A' subtahieren, und
        sub.b   #'A',d0         * zehn addieren
        add.b   #10,d0
        bra     hexein2         * nächstes nibbel holen

bstest2: cmp.b  #'F',d0         * gleich wie bstest1 für das
        bgt     hexeierr        * zweite Nibble
        cmp.b   #'A',d0
        blt     hexeierr
        sub.b   #'A',d0
        add.b   #10,d0
        bra     hexein4


************************************************************************
************************************************************************
* Variablen des Grundprogramms                                         *
************************************************************************
************************************************************************


********************************************************************
*  Menuedaten für das Hauptmenue, Adressen der Menutexte und        *
*  Adressen der Subroutines (haincjmp)                              *
********************************************************************


        data

haincjmp: dc.l  gotrack
        dc.l    gosync
        dc.l    gosektor
        dc.l    goclust
        dc.l    goformat
        dc.l    goinit
        dc.l    haupend2

menhaupt: dc.l  m1hau1a
        dc.l    m1hau1a1
        dc.l    m1hau1b
```

```
        dc.l    m1hau1b1
        dc.l    m1hau1c
        dc.l    m1hau1d1
        dc.l    m1hau1e

m1hau1a:  dc.b     '  TRACK  ',0
m1hau1a1: dc.b    'TRACK/SYNCS ',0
m1hau1b:  dc.b     ' SECTOR  ',0
m1hau1b1: dc.b     ' CLUSTER  ',0
m1hau1c:  dc.b     ' FORMAT  ',0
m1hau1d:  dc.b     '  FATS  ',0
m1hau1d1: dc.b    '  OPTIONS  ',0
m1hau1e:  dc.b     '  ENDE  ',0


hafrag1: dc.b     27,'p  A LITTLE DISK UTILITY   (C) U. Braun 1986  '
         dc.b     27,'q',0
hafrag2: dc.b     27,'p     DATA BECKER FLOPPY-BUCH FÜR ATARI ST     '
         dc.b     27,'q',0
hafrag3: dc.b     27,'p     Select Menue Items with Cursor-Keys      '
         dc.b     27,'q',0


**********************************************************************
*  Adressen der Sectormenutexte (mensect) und er Sectormenue-        *
*  routinen                                                          *
**********************************************************************


seincjmp: dc.l    incdrive
          dc.l    incside
          dc.l    inctrack
          dc.l    incsect
          dc.l    readsec
          dc.l    writsec
          dc.l    editsec
          dc.l    gohaupt

sedecjmp: dc.l    decdrive
          dc.l    decside
```

```
        dc.l    dectrack
        dc.l    decsect
        dc.l    readsec
        dc.l    writsec
        dc.l    editsec
        dc.l    gohaupt

mensect: dc.l   m1secta
        dc.l    m1sectb
        dc.l    m1sectc
        dc.l    m1sectd
        dc.l    m1secte
        dc.l    m1sectf
        dc.l    m1sectg
        dc.l    m1secth


m1secta: dc.b    ' drive: '
mdrive:  dc.b    '0',' ',0
m1sectb: dc.b    ' side: '
mside:   dc.b    '0',' ',0
m1sectc: dc.b    ' track: '
mtrack:  dc.b    '0','0',' ',0
m1sectd: dc.b    ' sector: '
msector: dc.b    '0','1',' ',0
m1secte: dc.b    '  READ  ',0
m1sectf: dc.b    '  WRITE  ',0
m1sectg: dc.b    '  EDIT  ',0
m1secth: dc.b    '  BACK  ',0

wrfrag1: dc.b    27,'p',' Write this Sector to: ',27,'q',0
wrfrag2: dc.b    27,'p  <yes,no> ? ',27,'q',0
wrfrag3: dc.b    27,'p Not written.  <press key> ',27,'q',0
sefrag1: dc.b    27,'p  SECTOR MODE  ',27,'q',0
edfrag1: dc.b    27,'p  EDIT MODE:  < return > := ENDE ',27,'q',0

*************************************************************************
*  Adressen für das Trackmenue                                          *
*************************************************************************
```

```
trincjmp: dc.l    incdrive
          dc.l    incside
          dc.l    inctrack
          dc.l    incstra
          dc.l    read1tr
          dc.l    writ1tr
          dc.l    edittr
          dc.l    gohaupt

trdecjmp: dc.l    decdrive
          dc.l    decside
          dc.l    dectrack
          dc.l    decstra
          dc.l    read1tr
          dc.l    writ1tr
          dc.l    edittr
          dc.l    gohaupt


mentrack: dc.l    m1secta
          dc.l    m1sectb
          dc.l    m1sectc
          dc.l    m1traca1
          dc.l    m1tracka
          dc.l    m1trackb
          dc.l    m1trackc
          dc.l    m1trackd


m1traca1: dc.b     ' Sec/Trac: '
setrack: dc.b     '0','9',' ',0
m1tracka: dc.b     ' READ ',0
m1trackb: dc.b     ' WRITE ',0
m1trackc: dc.b     ' EDIT Tr. ',0
m1trackd: dc.b     'BACK ',0

trfrag1: dc.b     27,'p  TRACK MODE  ',27,'q',0
trfrag2: dc.b    27,'p  TRACK WITH SYNCS MODE ',27,'q',0
trfrag3: dc.b     27,'p  Sector: ',0
```

```
trfrag4: dc.b     ' ',27,'q',0
trfrag5: dc.b     27,'p Write this Track to ',27,'q',0
trfrag6: dc.b     27,'p < yes/no > ',27,'q',0


****************************************************************************
*  Adressen für das Track mit Syncs-menue                                  *
****************************************************************************

syincjmp: dc.l    incdrive
          dc.l    incside
          dc.l    inctrack
          dc.l    rdtracks
          dc.l    readadr
          dc.l    gohaupt

sydecjmp: dc.l    decdrive
          dc.l    decside
          dc.l    dectrack
          dc.l    rdtracks
          dc.l    readadr
          dc.l    gohaupt


mensync:  dc.l    m1secta
          dc.l    m1sectb
          dc.l    m1sectc
          dc.l    m1synca
          dc.l    m1syncb
          dc.l    m1trackd

m1synca:  dc.b    ' READ WITH SYNCS ',0
m1syncb:  dc.b    ' ADDR. FIELD ',0


**************************************************************************
*   Cluster
**************************************************************************
```

```
clincjmp: dc.l    incdrive
        dc.l    incclust
        dc.l    rdclust
        dc.l    nextclst
        dc.l    wrclust
        dc.l    edclust
        dc.l    stclust
        dc.l    gohaupt

cldecjmp: dc.l    decdrive
        dc.l    decclust
        dc.l    rdclust
        dc.l    nextclst
        dc.l    wrclust
        dc.l    edclust
        dc.l    stclust
        dc.l    gohaupt

menclust: dc.l    m1secta
        dc.l    m1clusa
        dc.l    m1secte
        dc.l    m1clusb
        dc.l    m1sectf
        dc.l    m1clusd
        dc.l    m1clusc
        dc.l    m1secth


m1clusa: dc.b    ' CLUST: '
m1clusa1: dc.b   '0','0','0','0','  ',0
m1clusb: dc.b    '  NEXT  ',0
m1clusc: dc.b    ' STARTofFILE ',0
m1clusd: dc.b    ' EDIT ',0


clfrag1: dc.b    27,'p  CLUSTER MODE  ',27,'q',0
clfrag2: dc.b    27,'p  When leaving CLUSTER MODE, last read '
        dc.b    'Cluster is updatet in SECTOR Menue ',27,'q',0
clfrag4: dc.b    27,'p  This was the last Custer  ',27,'q',0
sclfrag1: dc.b   27,'p  Filename:          Fileattribut:  '
```

```
         dc.b     '   Startcluster:    Number of Bytes:  ',27,'q',0
sclfrag2: dc.b    27,'p  Start-Cluster mit <RETURN> ins Menue'
         dc.b     ' übernehmen, lesen durch <up>, <down>. ',27,'q',0
clfrag5: dc.b     27,'p Write this Cluster to: ',27,'q',0

trecsiz: dc.b     '  Bytes per Sector: ',0
tclsiz:  dc.b     '  Sector per Cluster: ',0
tclsizb: dc.b     '  Bytes per Cluster: ',0
trdlen:  dc.b     '  Sector per Directory: ',0
tfsiz:   dc.b     '  Sector per FAT: ',0
tfatrec: dc.b     '  Sektornumber second FAT:',0
tdatrec: dc.b     '  Sector of first Datecluster:',0
tnumcl:  dc.b     '  Number of clusters: ',0
tanzside: dc.b    '  Number of sides: ',0
tdir1:   dc.b     27,'p  First Directory-sektor on Side: 0  Track: 1 '
         dc.b     ' Sector: 3  ',27,'q',0
tdir2:   dc.b     27,'p  First Directory-sektor on Side: 1  Track: 0 '
         dc.b     ' Sector: 3  ',27,'q',0

tfolder: dc.b     '  Subdirectory  ',0
treadwr: dc.b     '  Read/Write    ',0
treadon: dc.b     '  Read only     ',0
thidden: dc.b     '  HIDDEN File   ',0
tdelet:  dc.b     '  Deleted       ',0
tdisname: dc.b    '  Diskettenname ',0


*********************************************************************
*  Format-Menue                                                     *
*********************************************************************

foincjmp: dc.l    incdrive
          dc.l    incside
          dc.l    inctrack
          dc.l    incstra
          dc.l    format1
          dc.l    xformat
          dc.l    gogaps
          dc.l    gohaupt
```

```
fodecjmp: dc.l      decdrive
          dc.l      decside
          dc.l      dectrack
          dc.l      decstra
          dc.l      format1
          dc.l      xformat
          dc.l      gogaps
          dc.l      gohaupt

formmen:  dc.l      m1secta
          dc.l      m1sectb
          dc.l      m1sectc
          dc.l      m1traca1
          dc.l      m1formd
          dc.l      m1forme
          dc.l      m1formf
          dc.l      m1formg

m1formd:  dc.b      ' FORMAT ',0
m1forme:  dc.b      ' XFORMAT ',0
m1formf:  dc.b      '  GAPS ',0
m1formg:  dc.b      '  BACK  ',0


fofrag1:  dc.b      27,'p  Format Track Mode  ',27,'q',0
fofrag2:  dc.b      27,'p  Track:',0
fofrag3:  dc.b      ' formatieren ?  <yes/no>  ',27,'q',0
fofrag4:  dc.b      27,'p  Nicht formatiert        <Taste> ',27,'q',0
fofrag5:  dc.b      '  auf Seite:',0
fofrag6:  dc.b      '  von Drive:',0
xffrag1:  dc.b      27,'p Wirklich mit neuen GAP's zwischen den '
          dc.b      'Sektoren formatieren? <yes/no> ',27,'q',0
xffrag2:  dc.b      27,'p Wait a second, then press key ',27,'q',0
m1form1:  dc.l      '  Format Track  ',0

**********************************************************************
*   Init Menue                                                       *
**********************************************************************
```

```
ininc jmp: dc.l    incdrive
           dc.l    incmaxtr
           dc.l    incmaxse
           dc.l    dodrivin
           dc.l    showbpb
           dc.l    gohaupt

indecjmp:  dc.l    decdrive
           dc.l    decmaxtr
           dc.l    decmaxse
           dc.l    dodrivin
           dc.l    showbpb
           dc.l    gohaupt

meninit:   dc.l    m1secta
           dc.l    m1drina
           dc.l    m1drinb
           dc.l    m1drinc
           dc.l    m1drinc1
           dc.l    m1drind


m1drina:   dc.b    '  MAXTRACK: '
max1tr:    dc.b    '7','9',' ',0
m1drinb:   dc.b    '  MAXSECTOR: '
max1se:    dc.b    '0','9',' ',0
m1drinc:   dc.b    '  INIT DRIVE  ',0
m1drinc1:  dc.b    '  SHOW BPB  ',0
m1drind:   dc.b    '  BACK  ',0

drifrag1:  dc.b    27,'p  INIT DRIVE MENUE  ',27,'q',0
drifrag2:  dc.b    27,'p  Bios Parameter Block of active drive    '
           dc.b    ' < press key >  ',27,'q',0
catfra1:   dc.b    27,'p   Directory starts at Side: 0 Track: 1 Sector: 3
'
           dc.b    27,'q',0
catfra2:   dc.b    27,'p   Directory starts at Side: 1 Track: 0 Sector: 3
'
           dc.b    27,'q',0
```

```
device:   dc.w    2
drive:    dc.w    0
side:     dc.w    0
track:    dc.w    0
sektor:   dc.w    0

seek:     dc.w    3
savesr:   dc.w    0
flstatus: dc.w   0

**********************************************************************
*  Gap-Menue                                                         *
**********************************************************************

gpincjmp: dc.l      incgap1
          dc.l      incgap2
          dc.l      incgap3
          dc.l      incgap4
          dc.l      incgap5
          dc.l      incbyte
          dc.l      goformat

gpdecjmp: dc.l      decgap1
          dc.l      decgap2
          dc.l      decgap3
          dc.l      decgap4
          dc.l      decgap5
          dc.l      decbyte
          dc.l      goformat

mengap:   dc.l      m1gapa
          dc.l      m1gapb
          dc.l      m1gapc
          dc.l      m1gapd
          dc.l      m1gape
          dc.l      m1gapf
          dc.l      m1gapg
```

```
m1gapa:  dc.b     ' GAP1: '
mgap1:   dc.b     '60 ',0
m1gapb:  dc.b     ' GAP2: '
mgap2:   dc.b     '12 ',0
m1gapc:  dc.b     ' GAP3: '
mgap3:   dc.b     '22 ',0
m1gapd:  dc.b     ' GAP4: '
mgap4:   dc.b     '40 ',0
m1gape:  dc.b     ' GAP5: '
mgap5:   dc.b     '664 ',0
m1gapf: dc.b     ' Byte/Sek: '
mdrisekt: dc.b    '0512 ',0
m1gapg:  dc.b     ' BACK ',0

drfrag1: dc.b     27,'p  Drive Format Mode  ',27,'q',0
gpfrag1: dc.b     27,'p  Change Gaps between Sektors  ',27,'q',0
slfrag1:  dc.b        27,'p   Bitte  eine  Sekunde  warten,  dann  Taste
',27,'q',0
slfrag3: dc.b     27,'p  SECTOR MODE  ',27,'q',0

drbyte:  dc.w     512
gap1:    dc.w     60
gap2:    dc.w     12
gap3:    dc.w     22
gap4:    dc.w     40
gap5:    dc.w     664

*************************************************************************

sadfrag1: dc.b    27,'p Track:  Seite:  Sektor:  Bytes:  Checsum(hex) '
          dc.b    27,'q',0

*************************************************************************
*  Hier stehen die Escape-Sequenzen für die Terminal-Emulation         *
*  wie: Revers ein- und ausschalten, Cursor positionieren ect.         *
*************************************************************************

clrest1: dc.b     27,'J',0
clrest2: dc.b     27,'K',0
```

```
revers1: dc.b      27,'p',0
revers2: dc.b      27,'q',0
loccurs1: dc.b     27,'Y',33,33,0
home1:    dc.b     27,'H',0
clear1:   dc.b     27,'E',0
curup1:   dc.b     27,'A',0
curdown1: dc.b     27,'B',0
insline1: dc.b     27,'L',0
delline1: dc.b     27,'l',0
overout1: dc.b     27,'w',0
curout1:  dc.b     27,'f',0
curon1:   dc.b     27,'e',0
spaces:   dc.b     '            ',0
hilcurs:  dc.b     27,'J',0

***********************************************************************
*  Adressen der Fehlertexte                                           *
***********************************************************************


errtab:   dc.l     error1
          dc.l     error2
          dc.l     error3
          dc.l     error4
          dc.l     error5
          dc.l     error6
          dc.l     error7
          dc.l     error8
          dc.l     error9
          dc.l     error10
          dc.l     error11
          dc.l     error12
          dc.l     error13
          dc.l     error14
          dc.l     error15
          dc.l     error16
          dc.l     error17
          dc.l     error18
          dc.l     error19
```

```
        dc.l    error20
        dc.l    error21
        dc.l    error22
        dc.l    error23
        dc.l    error24
        dc.l    error25
        dc.l    error26
        dc.l    error27
        dc.l    error28
        dc.l    error29

*************************************************************************
*  Hier folgen nun die eigentlichen Fehlertexte                         *
*************************************************************************

error1:  dc.b    27,'p',' NO BOOTSECTOR ',27,'q',0
error2:  dc.b    27,'p Directory-Sectors defect   <key> ',27,'q',0
error3:  dc.b    ' fehler3',0
error4:  dc.b    ' fehler4',0
error5:  dc.b    ' fehler5 ',0
error6:  dc.b    ' fehler6 ',0
error7:   dc.b     27,'p',' Diskette einlegen / Track nicht vorhanden
',27,'q',0
error8:  dc.b    ' fehler8 ',0
error9:  dc.b    27,'p',' Dieser Sektor existiert nicht !',27,'q',0
error10: dc.b    ' fehler10',0
error11: dc.b    ' fehler11',0
error12: dc.b    ' fehler12',0
error13: dc.b    ' fehler13 ',0
error14: dc.b    27,'p  Bitte Schreibschutz entfernen.  ',27,'q',0
error15: dc.b    ' fehler15 ',0
error16: dc.b    ' fehler16 ',0
error17: dc.b    ' fehler17',0
error18: dc.b    ' fehler18 ',0
error19: dc.b    ' fehler19 ',0
error20: dc.b   27,'p Kein weiterer Cluster ',27,'q',0
error21: dc.b    ' fehler21 ',0
error22: dc.b    ' fehler22 ',0
error23: dc.b    ' fehler23 ',0
error24: dc.b    ' fehler24 ',0
```

```
error25: dc.b     ' fehler25 ',0
error26: dc.b     ' fehler26 ',0
error27: dc.b     ' fehler27 ',0
error28: dc.b     ' fehler28 ',0
error29: dc.b     ' fehler29 ',0

******************************************************************************
pattern: dc.w     $ffff

         bss

menueadr: ds.l    1
ganz:     ds.l    1
revnum:   ds.l    1
jmptable: ds.l    1

wtrack:   ds.w    1
wsector:  ds.w    1
wside:    ds.w    1
wdrive:   ds.w    1
wclust:   ds.w    1

maxtrack: ds.w    1
maxsect:  ds.w    1
maxdriv:  ds.w    1
maxside:  ds.w    1
maxclust: ds.w    1

topptr:   ds.l    1
oldtop:   ds.l    1
botptr:   ds.l    1

spalte:   ds.w    1
zeile:    ds.w    1
head1:    ds.w    1
head2:    ds.w    1

curzei1:  ds.w    1
curspa1:  ds.w    1
oldzei1:  ds.w    1
```

```
oldspa1: ds.w    1

zeicount: ds.w   1
prcount: ds.w    1

retw1:   ds.w    1
incvar:  ds.l    1
decvar:  ds.l    1
usstack: ds.l    1
sustack: ds.l    1

dmastat: ds.w    1

currdma: ds.b    1
highdma: ds.b    1
middma:  ds.b    1
lowdma:  ds.b    1

maxhead: ds.w    1

savebpb: ds.l    1
recsiz:  ds.w    1
clsiz:   ds.w    1
clsizb:  ds.w    1
rdlen:   ds.w    1
fsiz:    ds.w    1
fatrec:  ds.w    1
datrec:  ds.w    1
numcl:   ds.w    1
bflags:  ds.w    1

oldsec:  ds.w    1

dflag:   ds.w    1
eflag:   ds.w    1
edflag:  ds.w    1

anzside: ds.w    1
tab1:    ds.w    1
```

```
oldclst: ds.w    1
newclst: ds.w    1
clstnum: ds.w    1
logsect: ds.w    1

asector: ds.w    1

topdma:  ds.l    1
editptr: ds.l    1

savereg: ds.l    16

maxdown: ds.w    1
maxup:   ds.w    1

lineavar: ds.l    1

varl1:   ds.l    1
varw1:   ds.w    1
varw2:   ds.w    1
varw3:   ds.w    1

dirptr:  ds.l    1

dirbuf:  ds.w    4000
fatbuf:  ds.w    4000
formbuf: ds.w    6000
platztr: ds.w    6000
         end
```

Hier folgen jetzt die vollwertigen Unterprogramme zu den einzelnen Menuepunkten die beim Listing "edit.s" noch auf rts endeten.

Am besten halten Sie sich an die von mir vorgegebene Reihenfolge, da manche Menuepunkte auf Unterprogramme anderer Menues zurückgreifen. Wenn Sie das Programm in der hier vorgeschlagenen Weise vervollständigen ergeben sich keine Probleme dieser Art.

Zuerst die Unterprogramme des Options-Menues, damit Sie den Maximal-Track und Sektor bestimmen und außerdem den Bios-Parameterblock ansehen können.

```
***************************************************************************
***************************************************************************
*  Subroutines des Menuepunktes OPTION, sollten zuerst implementiert      *
*  werden, da hiermit auf die Möglichkeit gegeben wird den 10. Sektor     *
*  auf dem 82 Track ect. zu lesen, außerdem werden einige Routinen        *
*  von anderen Programmteilen aufgerufen                                  *
***************************************************************************
***************************************************************************


***************************************************************************
*  aktuelles Drive (aus dem Menue) initialisieren, und die Variablen     *
*  des Biosparameterblockes speichern                                     *
***************************************************************************

initdriv: move.w wdrive,d0       * aktuelles drive
          move.w  wdrive,-(a7)   * auf den Stack
          move.w  #7,-(a7)       * Getbpb Funktion
          trap    #13            * BIOS-Trap
          addq.l  #4,a7          * Stack restaurieren
          tst.l   d0             * Fehler aufgetreten?
          bne     doinit1
          move.w  d0,-(a7)       * wenn ja, dann diesen übergeben
          jsr     errhand        * und dann zurück
          bra     doiniten
doinit1:  move.l  d0,a0          * sonst d0 = Basisadresse vom BPB
          move.w  (a0)+,recsiz   * Bytes pro Sector
          move.w  (a0)+,clsiz    * Sectors/ Cluster
          move.w  (a0)+,clsizb   * Bytes/Cluster
          move.w  (a0)+,rdlen    * Sectors/Directory
          move.w  (a0)+,fsiz     * Sectors/Fat
          move.w  (a0)+,fatrec   * Sec. Num. des zweiten FAT
          move.w  (a0)+,datrec   * Sec.Num. des ersten Dat.CLust.
```

```
        move.w  (a0)+,numcl    * Anzahl der DAtencluster
        move.w  (a0)+,bflags   * Flags
        move.w  (a0)+,anzside  * noch dummy
        move.w  (a0)+,anzside  * Anzahl der Seiten
doiniten: rts                  * und zurück


***********************************************************************
* liest die Fatsektoren von der Diskette in den Fatbuffer            *
***********************************************************************

rdfat:  move.w  wdrive,-(a7)   * aktuelles drive
        move.w  fatrec,-(a7)   * Sectnummer des zweiten Fat
        move.w  fsiz,-(a7)     * Anzahl der Sektoren pro FAT
        move.l  #fatbuf,-(a7)  * Pufferadresse auf Stack
        move.w  #2,-(a7)       * unbedingt lesen
        move.w  #4,-(a7)       * Rwabs Funktion
        trap    #13            * BIOS-Trap
        add.l   #14,a7         * Stack restaurieren
        tst.w   d0             * Fehler aufgetreten?
        bmi     rdfater        * wenn ja, dann handlen
rdfatend: rts                  * sonst zurück

rdfater: move.w d0,-(a7)       * Fehlernummer auf Stack
        jsr     errhand        * handlen
        bra     rdfatend       * und zurück


***********************************************************************
*  liest die Directory-Sektoren von der Diskette in einen Puffer     *
***********************************************************************

rddir:  move.w  wdrive,-(a7)   * aktuelles Drive
        move.w  fsiz,d0        * Anzahl der Fatsektoren
        lsl.w   #1,d0          * mal zwei (FAT's) plus eins
        addq.w  #1,d0          * gleich logische Sektornummer des
        move.w  d0,-(a7)       * ersten Directorysektors
        move.w  rdlen,-(a7)    * Anzahl Directory-Sektoren
        move.l  #dirbuf,-(a7)  * Adresse des Puffers
```

```
         move.w   #2,-(a7)          * unbedingt lesen
         move.w   #4,-(a7)          * Rwabs Funktion
         trap     #13               * BIOS
         add.l    #14,a7
         tst.w    d0                * Fehler?
         bmi      rddirer           * ja
rddirend: rts                       * wenn nicht, dann sofort zurück

rddirer: move.w   d0,-(a7)          * Fehlernummer
         jsr      errhand
         bra      rddirend


************************************************************************
* erhöht die maximale Anzahl der einstellbaren Tracks im Init-Drive-  *
* Menue, bis zu diesem Wert kann man dann in allen anderen Menues      *
* die aktuelle Tracknummer erhöhen                                     *
************************************************************************


incmaxtr: move.w  maxtrack,d0
         cmp.w    #99,d0            * 99 ist das Maximum
         blt      incma1            * sonst das gleiche Verfahren wie
         move.w   #0,d0             * bei den bisherigen Menuechanges
         bra      incma2

incma1:  addq.w   #1,d0
incma2:  move.w   d0,maxtrack
         ext.l    d0
         divu     #10,d0
         add.b    #'0',d0
         move.b   d0,max1tr         * auch im Menuetext ändern
         swap     d0
         add.b    #'0',d0
         move.b   d0,max1tr+1
         jsr      dispmen           * Menue anzeigen
         rts                        * und zurück
```

```
decmaxtr: move.w  maxtrack,d0      * erniedrigt die maximal einstellbare
          cmp.w   #0,d0            * Tracknummer
          ble     decma1
          subq.w  #1,d0
          bra     decma2
decma1:   move.w  #99,d0
decma2:   move.w  d0,maxtrack
          ext.l   d0
          divu    #10,d0
          add.b   #'0',d0
          move.b  d0,max1tr        * im Menuetext ändern
          swap    d0
          add.b   #'0',d0
          move.b  d0,max1tr+1
          jsr     dispmen          * Menue anzeigen
          rts                      * und zurück

incmaxse: move.w  maxsect,d0       * nun das gleiche mit dem maximal
          cmp.w   #99,d0           * einstellbaren Sektor
          blt     incmas1
          move.w  #0,d0
          bra     incmas2

incmas1:  addq.w  #1,d0
incmas2:  move.w  d0,maxsect
          ext.l   d0
          divu    #10,d0
          add.b   #'0',d0
          move.b  d0,max1se        * im Menuetext einstellen
          swap    d0
          add.b   #'0',d0
          move.b  d0,max1se+1
          jsr     dispmen          * Menue anzeigen
          rts                      * und zurück

decmaxse: move.w  maxsect,d0       * erniedrig den maximal einstell-
          cmp.w   #0,d0            * baren Sektor
          ble     decmas1
```

```
        subq.w   #1,d0
        bra      decmas2
decmas1: move.w   #99,d0
decmas2: move.w  d0,maxsect
        ext.l    d0
        divu     #10,d0
        add.b    #'0',d0
        move.b   d0,max1se      * im Menuetext ändern
        swap     d0
        add.b    #'0',d0
        move.b   d0,max1se+1
        jsr      dispmen        * Menue anzeigen
        rts                     * und zurück


*************************************************************************
*  hier folgt die eigentliche Drive-Init-routine, die sowohl das        *
*  aktuelle Drive (in wdrive) initialisiert und den Bios-Parameter-     *
*  Block einliest als auch die FAT- und Directory-Sektoren in die       *
*  jeweiligen Puffer einliest                                           *
*************************************************************************

dodrivin: jsr     initdriv       * Drive initialisieren
        jsr      rdfat          * FAT Sektoren einlesen
        jsr      rddir          * Directory Sektoren einlesen
        jsr      showbpb        * BIOS-Parameter-Block anzeigen
        jsr      curlinks
dodriven: rts                   * und zurück


*************************************************************************
* Anzeigen des BIOS-Parameter-Blockes                                   *
*************************************************************************

showbpb: move.w   #4,zeile       * Cursor in Zeile 4, Spalte 10
        move.w   #10,spalte
        jsr      loccurs        * positionieren
        move.l   #drifrag2,a0   * Message ausgeben
```

```
        jsr     printf
        move.w  #42,tab1        * Tabpunkt auf dem Bildschirm für die
        move.w  #6,zeile        * Ausgabe der Zahlen
        move.w  #12,spalte
        jsr     loccurs
        move.l  #trecsiz,a0     * Bytes pro Cluster
        jsr     printf
        jsr     curstab         * Text schreiben
        move.w  recsiz,-(a7)    * Bytes pro Cluster als Dezimalzahl
        jsr     dezpr           * schreiben
        addq.w  #1,zeile        * eine Zeile weiterschalten
        move.w  #12,spalte
        jsr     loccurs
        move.l  #tclsiz,a0      * Sektoren pro Cluster
        jsr     printf
        jsr     curstab
        move.w  clsiz,-(a7)
        jsr     dezpr
        addq.w  #1,zeile
        move.w  #12,spalte
        jsr     loccurs
        move.l  #tclsizb,a0     * Bytes pro Cluster
        jsr     printf
        jsr     curstab
        move.w  clsizb,-(a7)
        jsr     dezpr
        addq.w  #1,zeile
        move.w  #12,spalte
        jsr     loccurs
        move.l  #trdlen,a0      * Sektoren pro Directory
        jsr     printf
        jsr     curstab
        move.w  rdlen,-(a7)
        jsr     dezpr
        addq.w  #1,zeile
        move.w  #12,spalte
        jsr     loccurs
        move.l  #tfsiz,a0       * Sektoren pro FAT
        jsr     printf
        jsr     curstab
```

```
        move.w  fsiz,-(a7)
        jsr     dezpr
        addq.w  #1,zeile
        move.w  #12,spalte
        jsr     loccurs
        move.l  #tfatrec,a0     * Sektornummer des zweiten FAT
        jsr     printf
        jsr     curstab
        move.w  fatrec,-(a7)    *
        jsr     dezpr
        addq.w  #1,zeile
        move.w  #12,spalte
        jsr     loccurs
        move.l  #tdatrec,a0     * Sektornummer des ersten Daten-
        jsr     printf          * clusters
        jsr     curstab
        move.w  datrec,-(a7)
        jsr     dezpr
        addq.w  #1,zeile
        move.w  #12,spalte
        jsr     loccurs
        move.l  #tnumcl,a0      * Anzahl der Datencluster
        jsr     printf
        jsr     curstab
        move.w  numcl,-(a7)
        jsr     dezpr
        addq.w  #1,zeile
        move.w  #12,spalte
        jsr     loccurs
        move.l  #tanzside,a0    * Anzahl der Diskettenseiten
        jsr     printf
        jsr     curstab
        move.w  anzside,-(a7)
        jsr     dezpr
        addq.w  #2,zeile
        move.w  #10,spalte
        jsr     loccurs
        move.l  #tdir1,a0       * Angabe wo sich der erste Directory-
        move.w  anzside,d0      * Sektor befindet, differiert bei
        cmp.w   #2,d0           * ein- und zweiseitigen Disketten
```

```
        bne       showbpb1
        move.l    #tdir2,a0
showbpb1: jsr     printf
        jsr       leerebuf         * Tastaturpuffer leeren
        jsr       wtast            * auf Tastendruck warten
        jsr       cursmess
        jsr       delline
        jsr       cursmess
        move.l    #drifrag1,a0     * Message anzeigen
        jsr       printf
        rts                        * und zurück
```

Nun die Unterprogramme des Track-Menues

```
**********************************************************************
**********************************************************************
*                                                                    *
*  Unterprogramme des TRACK Menues plus eigene Sektorschreibroutine  *
*                                                                    *
**********************************************************************
**********************************************************************


**********************************************************************
*  eigene Sektorschreibroutine, greift direkt auf Controller und DMA *
*  chip zu. die XBIOS-Routine zum Sektorschreiben läßt sich im Gegen- *
*  satz zur Sektorleseroutine nicht dazu "überreden" Sektoren mit    *
*  1024 Byte zu schreiben, daher wird diese Routine aufgerufen.      *
*  Damit diese Funktion eingefügt werden kann, muß zuerst das        *
*  rdstrack-Menue implementiert werden, da einige Routinen dieses    *
*  Meneus aufgerufen werden (super, seldrive, ect.) In der           *
*  Grundaustattung des Programms (nur mit Sektormenue) ist es        *
*  leider nicht möglich Sektoren mit 1024 Bytes pro Sektor zu        *
*  schreiben.                                                        *
**********************************************************************

selfsect: jsr     super            * Supervisor-Mode einschalten
        st        flock            * Floppy-Interrupt ausschalten
```

```
        jsr      seldrive          * drive und Seite selectieren
        jsr      flreset           * reset des Controllers
        jsr      searcht           * Track in wtrack suchen
        jsr      selwrite          * Sektor schreiben
        sf       flock             * Floppy-Interrupt wieder zulassen
        jsr      leerebuf          * Tastaturpuffer leeren
        jsr      cursmess          * Cursor positionieren
        jsr      delline           * Zeile löschen
        jsr      flreset           * Controller reset
        jsr      user              * User-Mode einschalten
        move.l   #slfrag1,a0       * Message ausgeben
        jsr      printf
        jsr      wtast             * auf Tastendruck warten
        jsr      super             * Super-Visor-Mode einschalten
        jsr      deselect          * Floppy deselektieren
        jsr      user              * User-Mode einschalten
        jsr      cursmess          * Cursor positionieren
        jsr      delline           * Zeile löschen
        jsr      cursmess          * wieder positionieren
        move.l   #slfrag3,a0       * Message ausgeben
        jsr      printf
        movem.l  (a7)+,a3-a6/d3-d7 * Register zurückholen
        rts                        * und zurück


************************************************************************
*                                                                      *
*  Hier wird nun der Sektor auf die Diskette geschrieben.              *
*                                                                      *
************************************************************************

selwrite: jsr    setplatz          * Adresse des Puffers einstellen
         move.w #$190,dmamode      * Auf Schreiben umschalten
        move.w   #$90,dmamode      * durch "Toggeln" der Schreib-
        move.w   #$190,dmamode     * Lese-Leitung
        move.w   #4,d6             * Sectorcount-Register mit 4
        jsr      wrcontr           * beschreiben
        move.w   #$184,dmamode     * Sectorregister des FDC anwählen
        move.w   wsector,d6        * aktuellen Sektor an FDC übergeben
```

```
        jsr      wrcontr
        move.w   #$180,dmamode       * Controller anwählen
        move.w   #$a0,d6             * Sektor-Write-Befehl an Controller
        jsr      wrcontr             * übergeben
        move.l   #$50000,d7          * Time-out-Zähler
selwrit1: btst   #5,mfp              * Interrupt-Eingang des FDC am MFP
        beq      selwrend            * wenn 1 dann fertig
        subq.l   #1,d7               * Timeout Zähler erniedrigen, wenn
        bne      selwrit1          * noch nicht abgelaufen, weiterwarten
        move.w   #-9,-(a7)           * Sonst Fehlernummer 9 auf Stack
        jsr      errhand             * an Errorhandler übergeben
        jsr      cursmess            * anschließend Ausgabezeile
        jsr      delline             * löschen
        rts

selwrend: jsr    rdstatus            * hierhin wird bei fehlerfreiem Ab-
        move.w   flstatus,d0         * lauf verzweigt
        btst     #6,d0               * writeprotect
        bne      selwerr1            * ja
        rts                          * wenn nicht, dann zurück


selwerr1: move.w  #-8,-(a7)         * Fehlermeldung Nr. 8(writeprotect)
        jsr      errhand            * ausgeben und anschließend Ausgabe-
        jsr      cursmess           * zeile löschen
        jsr      delline
        rts


*************************************************************************
*************************************************************************
*  Subroutines des Menuepunktes TRACK des Hauptmenues                   *
*************************************************************************
*************************************************************************

*************************************************************************
*                                                                       *
*  Diese Routine ließt einen ganzen Track, bzw. die in der Variablen    *
*  asector übergebene Anzahl von Sektoren. Es wird von Standard-        *
```

```
*  sektoren mit 512-Byte ausgegangen, so daß eventuelle Abweichungen *
*  durch Verändern der Variablen asector ausgeglichen werden müssen   *
*                                                                     *
***********************************************************************


read1tr: move.w   #512,d0          * Standardsektorgröße
         mulu     asector,d0       * Anzahl der Sektoren pro Track
         move.w   d0,maxhead       * Maximale Anzahl Bytes als Zähler
         move.w   asector,-(a7)    * Anzahl Sektoren/Track aus Menue
         move.w   wside,-(a7)      * aktuelle Seite
         move.w   wtrack,-(a7)     * aktueller Track
         move.w   #1,-(a7)         * ab Sektor eins
         move.w   wdrive,-(a7)     * aktuelles Drive
         clr.l    -(a7)            * Dummy-Langwort
         move.l   #platztr,-(a7)   * Pufferadresse
         move.w   #8,-(a7)         * X-tended Bios Funktion 8
         trap     #14              * aufrufen, anschließend den Stack
         add.l    #20,a7           * bereinigen und auf fehlerfreien
         tst.w    d0               * Ausgang untersuchen
         bmi      readt1er         * Wenn Fehler, dann ausgeben
         jsr      showtr           * sonst den Track anzeigen und
readt12: rts                       * anschließend zurück

readt1er: move.w  d0,-(a7)         * Fehlernummer auf dem Stack an den
         jsr      errhand          * Fehlerhändler übergeben, an-
         jsr      leerebuf         * schließend Tastaturpuffer leeren
         jsr      wtast            * und auf Tastendruck warten
         jsr      cursmess         * Message ausgeben
         move.l   #trfrag1,a0
         jsr      printf
         jsr      delrest
         bra      readt12          * und zurück


***********************************************************************
* erhöht bei Cursor-up Betätigung die Anzahl der Sektoren pro Track   *
* im Menue                                                            *
***********************************************************************
```

```
incstra: move.w  asector,d0      * Anzahl der Sektoren pro Track
         cmp.w   maxsect,d0      * mit der Maximalanzahl derSektoren
         blt     incst1          * vergleichen, wenn größer oder
         move.w  #0,d0           * gleich, dann Anzahl Sektoren/Track
         bra     incst2          * auf Null, und zurück
incst1:  addq.w  #1,d0           * sonst eins zu Anzahl Sekt./Tr.
incst2:  move.w  d0,asector      * addieren
         ext.l   d0              * Die Änderung auch im Menuetext
         divu    #10,d0          * eintragen, durch Divison durch
         add.b   #'0',d0         * 10 in einzelne ASCII-Byte zerlegen
         move.b  d0,setrack      * und ins Menue eintragen
         swap    d0              * auch das Low-Byte richtig
         add.b   #'0',d0         * ins ASCII-Format umwandeln
         move.b  d0,setrack+1    * und ins Menue eintragen, an-
         jsr     dispmen         * schließend das Menue anzeigen
         rts                     * und zurück

***********************************************************************
*  erniedrigt sectoren pro track im menue                             *
***********************************************************************

decstra:  move.w   asector,d0      * sektoren/track
          cmp.w    #0,d0
          ble      decst1          * wenn größer als null, dann eins
          subq.w   #1,d0           * subtrahieren, sonst
          bra      decst2
decst1:   move.w   maxsect,d0      * Maximalanzahl einsetzen
decst2:   move.w   d0,asector
          ext.l    d0
          divu     #10,d0
          add.b    #'0',d0
          move.b   d0,setrack      * ins Menueeintragen
          swap     d0
          add.b    #'0',d0
          move.b   d0,setrack+1
```

```
        jsr       dispmen            * Menue anzeigen und zurück
        rts

****************************************************************************
*  versorgt die Variablen der allgemeinen Editroutine mit Werten,          *
*  (maxdown, maxup ect.) und ruft die Editroutine dann auf                 *
****************************************************************************

edittr: move.w    #0,maxdown         * es sollen jeweils nur 512 Byte
        move.w    #208,maxup         * editiert werden,
        move.w    #18,zeicount       * und 19 Zeilen werden angezeigt
        move.l    topptr,d0          * Zeiger in den Trackpuffer
        sub.l     #platztr,d0        * minus Anfangsadresse des Puffers
        divu      #512,d0            * durch die Anzahl der Bytes pro
        swap      d0                 * Sektor dividieren, wenn ein Rest
        tst.w     d0                 * vorhanden, dann war es nicht der
        beq       edittr1            * der Anfang eines Sektors und es
        sub.l     #256,topptr        * muß 256 subtrahiert werden
edittr1: move.l   topptr,editptr     * diesen Zeiger auf Sektoranfang
        move.w    #0,head2           * im Trackpuffer an editit über
        move.w    #20,spalte         * geben, Message in Spalte 20 der
        move.w    #2,zeile           * Zeile 2 ausgeben
        jsr       loccurs
        move.l    #edfrag1,a0
        jsr       printf
        jsr       editit             * und edit aufrufen
        jsr       cursmess           * anschließend Messagezeile
        jsr       delline            * löschen und
        jsr       cursmess
        move.l    #trfrag1,a0        *
        jsr       printf
        jsr       curlinks           * Menue wieder auf read einstellen
        jsr       curlinks
        jsr       cursbuf            *
        jsr       clrest             * restlichen Bildschirm löschen
        rts                          * und zurück
```

```
*************************************************************************
*  ermöglicht das Ansehen des gesamten, in den Puffer eingelesenen     *
*  Tracks                                                              *
*************************************************************************

showtr: move.w #0,head2         * Byte-Zähler
        move.l  #platztr,topptr * Pufferanfang
        move.w  #0,edflag       * Flag
        move.w  #15,zeicount    * 16 Zeilen sollen jeweils gezeigt
        move.w  #2,zeile        * werden, in Spalte 59 der zweiten
        move.w  #59,spalte      * Zeile den aktuellen Sektor
        jsr     loccurs
        move.l  #trfrag3,a0     * anzeigen
        jsr     printf
        clr.w   d0
        move.w  #1,d0
        move.w  d0,-(a7)
        jsr     dezpr           * Sektor drucken
        move.l  #trfrag4,a0
        jsr     printf

showt1: move.w  #4,zeile        * Cursor positionieren
        move.w  #0,spalte
        jsr     loccurs
        jsr     clrest          * Rest des Bildschirms löschen
        jsr     leerebuf
showt2: jsr     dispbuf         * die erste Seite anzeigen und
showt3: jsr     taste           * Tastatur abfragen
        swap    d0
        cmp.b   #$48,d0         * Cursor up ?
        beq     showtup
        cmp.b   #$50,d0         * Cursor down ?
        beq     showtdo
        cmp.b   #$1c,d0         * Return ?
        beq     showten1
        cmp.b   #$4b,d0         * Cursor links ?
        beq     showtli
        cmp.b   #$4d,d0         * Cursor rechts ?
        beq     showtre
```

```
         bra       showt3              * keins von allem weiter abfragen
showtre: jsr       currecht            * cursor auf rechten Menuepunkt
         bra       showten1            * und zurück

showtli: jsr       curlinks            * linken Menuepunkt invers
         bra       showten1            * darstellen und zurück

showtup: move.w    head2,d0            * Byte-Zähler mit
         cmp.w     #0,d0               * Null vergleichen
         beq       showtuen            * wenn nicht gleich null,
         sub.w     #256,head2          * dann 256, entspricht einem halben
         sub.l     #256,topptr         * Sektor, abziehen
showtuen: move.w   head2,d0            * Byte-Zähler
         lsr.w     #8,d0               * dividiert durch 512
         lsr.w     #1,d0
         add.w     #1,d0               * plus eins gleich Sektornummer
         move.w    d0,varw3
         move.w    #59,spalte
         move.w    #2,zeile
         jsr       loccurs
         move.l    #trfrag3,a0         * die aktuelle Sektornummer in
         jsr       printf              * Zeile 2 ausgeben
         move.w    varw3,-(a7)
         jsr       dezpr
         move.l    #trfrag4,a0
         jsr       printf
         jsr       delrest             * den Rest der Zeile löschen

showtue1:  bra     showt2              * und zur Schleife zurück

**********************************************************************

showtdo: move.w    head2,d0            * Cursordown-Handling
         move.w    maxhead,d1
         sub.w     #256,d1
         cmp.w     d1,d0
         beq       shwtrden
         add.w     #256,head2          * 256 zu Pufferpointer und
         add.l     #256,topptr         * und Byte-Zähler addieren
```

```
shwtrden: move.w  head2,d0
          lsr.w   #8,d0
          lsr.w   #1,d0          * den Byte-Zähler durch 512
          add.w   #1,d0          * dividieren und eins addieren
          move.w  d0,varw3       * ergibt aktuelle Sektornummer
          move.w  #59,spalte
          move.w  #2,zeile
          jsr     loccurs
          move.l  #trfrag3,a0
          jsr     printf
          move.w  varw3,-(a7)    * die Sektornummer anzeigen
          jsr     dezpr
          move.l  #trfrag4,a0
          jsr     printf
          jsr     delrest        * Rest der Zeile löschen
shwtrd1:  bra     showt2
showten1: jsr     leerebuf       * Tastaturpuffer leeren
          rts                    * und zurück


*******************************************************************


writ1tr:  move.l  a4,-(a7)       * schreibt den eingelesenen Track
          move.w  #2,zeile       * wieder auf die Diskette
          jsr     loccurs
          jsr     delline
          move.l  #trfrag5,a0
          jsr     printf
          move.w  #33,d2         * 34 Byte ab m1secta auf Bildschirm
          move.l  #m1secta,a4    * ausgeben
writ1t1:  move.b  (a4)+,d0
          move.w  d0,-(a7)
          jsr     conout
          dbra    d2,writ1t1
          move.l  #trfrag6,a0    * Fragen ob wirklich auf Diskette
          jsr     printf         * geschrieben werden soll
          jsr     leerebuf       * Tastaturpuffer leeren
          jsr     wtast          * Tastatur abfragen und auf
          cmp.b   #'Y',d0        * kleines und großes Ypsilon über
```

```
         beq        writ1t2          * prüfen
         cmp.b      #'y',d0
         bne        writ1ten         * wenn andere Taste, dann nicht
writ1t2: move.w     asector,-(a7)    * Anzahl der Sektoren auf Stack
         move.w     wside,-(a7)      * aktuelle Seite
         move.w     wtrack,-(a7)     * aktueller Track
         move.w     #1,-(a7)         * Startsektor gleich eins
         move.w     wdrive,-(a7)     * aktuelles Drive
         clr.l      -(a7)            * Dummy-Langwort
         move.l     #platztr,-(a7)   * Pufferadresse
         move.w     #9,-(a7)         * Flopwr Befehl auf Stack
         trap       #14              * XBIOS-Trap
         add.l      #20,a7           * Stack restaurieren
         tst.w      d0               * ist ein Fehler aufgetreten ?
         bmi        writ1er1         * ja

writ1ten: jsr       cursmess         * kein Fehler, dann Statuszeile
         jsr        delline          * löschen und Message ausgeben
writ1te1: jsr        cursmess
         move.l     #trfrag1,a0
         jsr        printf
         move.l     (a7)+,a4         * holt a4 zurück
         rts

writ1er1: move.w    d0,-(a7)         * Fehlernummer auf den Stack
         jsr        errhand          * Fehler handeln
         bra        writ1te1         * und zum Ende
```

Nun die Unterprogramme des Track with Sync-Menues

```
***********************************************************************
***********************************************************************
*  Subroutines des Menuepunktes TRACK with SYNCS, die Routinen        *
*  greifen auf keine anderen Routinen zurück, kann daher wahlfrei     *
*  implementiert werden                                               *
***********************************************************************
***********************************************************************
```

```
*********************************************************************
*  zuerst einige häufig benutzte Variablen                          *
*********************************************************************


dmamode: equ    $ff8606
dmadat:  equ    $ff8604

dmahigh: equ    $ff8609
dmamid:  equ    $ff860b
dmalow:  equ    $ff860d

mfp:     equ    $fffa01

flselec: equ    $ff8800
flwrite: equ    $ff8802

flock:   equ    $43e


*********************************************************************
*  schaltet den Prozessor in den Supervisor-Mode, sollte sich der   *
*  Prozessor schon im Supervisormode befinden, passiert nichts      *
*********************************************************************

super:   move.l   #1,-(a7)      *
         move.w   #$20,-(a7)    * GEMDOS-Funktion Super
         trap     #1            * Test ob schon im Super-Modus
         add.l    #6,a7
         tst.w    d0            *
         bne      super1        * Prozessor schon im Super-Mode
         clr.l    -(a7)         * wenn nicht, dann auf Super-
         move.w   #$20,-(a7)    * visor-Mode umschalten
         trap     #1
         add.l    #6,a7
         move.l   d0,usstack    * Userstack speichern
super1:  rts                    * ab hier im Supervisor-Mode
```

```
***************************************************************************
*  schaltet wieder zurück in den User-Mode                                *
***************************************************************************

user:    move.l   #1,-(a7)
         move.w   #$20,-(a7)   * GEMDOS-Funktion Super
         trap     #1
         add.l    #6,a7
         tst.w    d0
         beq      user1        * schon im User-Mode
         move.l   usstack,-(a7)
         move.w   #$20,-(a7)
         trap     #1
         add.l    #6,a7
user1:   rts

***************************************************************************

fwarten: dbra     d7,fwarten
         rts


***************************************************************************
* Reset des Floppydiskcontrollers (FDC)                                   *
***************************************************************************

flreset: jsr      super        * in Supervisor-Mode umschalten
         move.w   #$80,dmamode * Zugriff auf FDC-Register
         move.w   #$d0,d6      * Reset durch Interupt-Befehl
         jsr      wrcontr      * Befehl an Controller
         move.w   #40,d7       * ein wenig warten
         jsr      fwarten
         rts                   * und zurück
```

```
************************************************************************
*  liest das Statusrgister des Controllers und speichert dieses       *
************************************************************************


rdcontr: jsr      super         * Supervisor-Mode einschalten
         move.w   dmadat,d3     * Statusregister nach D3
         jsr      readco1       * ein wenig warten
readco1: move.w   sr,-(a7)
         move.w   d7,-(a7)      * Timout-Zähler retten
         move.w   #40,d7
readco2: dbra     d7,readco2
         move.w   (a7)+,d7      * anschließend zurück
         move.w   (a7)+,sr
         rts

************************************************************************
*  übergibt die in D6 stehenden Zahl an den Floppy-Disk-Controller    *
************************************************************************

wrcontr: jsr      super         * Supervisor on
         jsr      readco1
         move.w   d6,dmadat
         jsr      readco1       *ein wenig warten
         rts

************************************************************************
*  liest das Statusregister des FDC und speichert dieses in flstatus  *
************************************************************************

rdstatus: jsr     super         * Supervisor on
         jsr      readco1
         move.w   dmadat,flstatus  * Status nach flstatus
         jsr      readco1       * ein wenig warten und
         rts                    * dann zurück
```

```
****************************************************************************
*  selektiert das aktuelle Drive (rote Lampe brennt)                      *
****************************************************************************

seldrive: jsr     super          * Supervisor on
          move.w  wdrive,d0      * aktuelles Drive
          cmp.w   #1,d0          * größer als 1
          bgt     seldrend       * wenn ja, dann zuück
          addq.b  #1,d0          * sonst mit aktueller Seite
          lsl.b   #1,d0
          or.w    wside,d0       * verknüpfen,
          eor.b   #7,d0
          and.b   #7,d0
select:   move.w  sr,-(a7)
          or.w    #$700,sr       * Interrupt ausschalten, da derFloppy
          move.b  #$e,flselec    * Interrupt die Drives wieder
          move.b  flselec,d1     * deselctiert
          and.b   #$f8,d1
          or.b    d0,d1
          move.b  d1,flwrite     * an ACIA übergeben
          move.w  (a7)+,sr       * Staturregister zurückholen
seldrend: rts                    * und zurück


****************************************************************************
*  aktuelles Drive deselektieren (rote Lampe erlischt)                    *
*  das Timing zwischen Floppy-Reset und deselektieren muß stimmen,        *
*  sonst läuft der Floppy-Motor weiter                                    *
****************************************************************************

deselect: jsr     super          * Supervisor on
          move.w  #$80,dmamode   * FDC-Register auswählen
          move.b  #7,d0
          jsr     select         * deselektieren
          rts                    * und zurück
```

```
***********************************************************************
*  liest einen ganzen Track mit allen Syncs in den Puffer der bei    *
*  Adresse platztr beginnt                                            *
***********************************************************************


rdstrack: jsr     super           * Supervisor on
          clr.l   currdma
          move.w  sr,varw3        * altes Statusregister retten
          move.w  #$2700,sr       * Interrupts sperren, ist eigentlich
          move.w  #$90,dmamode    * nicht nötig, Sektor-Count Register
          move.w  #$190,dmamode   * Toggle DMAMODE um auf Lesen umzu-
          move.w  #$90,dmamode    * schalten, und das DMA-Register zu
          move.w  #$16,d6         * löschen, es sollen 22*512 Byte
          move.w  #512,d2         * gelesen werden, (soviele stehen
          mulu    d6,d2           * gar nicht auf der Diskette)
          move.w  d2,maxhead      * aber
          add.l   #platztr,d2     * errechnen der DMA-Endadresse
          move.l  d2,topdma       * diese speichern
          jsr     wrcontr         * d6 (Anzahl der Sektoren) an FDC
          move.l  #platztr,d0     * Adresse des DMA-Puffers an
          move.b  d0,dmalow       * DMA-Chip übergeben
          lsr.l   #8,d0
          move.b  d0,dmamid
          lsr.l   #8,d0
          move.b  d0,dmahigh
          move.w  #$80,dmamode    * FDC-Register anwählen
          move.w  #$e8,d6         * Readtrack-Command an FDC
          jsr     wrcontr         * übergeben
          move.l  #$50000,d7      * Timeout-Zähler
          move.l  topdma,a5       * DMA-Endadresse
          move.w  #$200,d0        * ein wenig waren
rd1:      dbra    d0,rd1

rdstrl1:  btst    #5,mfp          * Befehl schon abgearbeitet?
          beq     rdtrend1        * wenn ja, dann ende
          subq.l  #1,d7           * Sonst Time-out-Zähler erniedr.
          beq     rdtrerr1        * wenn Zähler abgelaufen, dann Fehler
          move.b  dmahigh,highdma  * Testen ob End DMA-Adresse schon
          move.b  dmamid,middma   * erreicht, ist unnötig, da der
```

```
        move.b  dmalow,lowdma   * Controller vorher abbricht (weniger
        cmp.l   currdma,a5      * Byte auf Diskette)
        bgt     rdstrl1

rdtrend1: move.w #$90,dmamode   * auf Sektorcountregister umschalten
        move.w  dmamode,d5      * Status des DMA-Chips lesen
        move.w  d5,dmastat      * und speichern
        btst    #0,d5
        beq     rdtrerr2
        move.w  #$80,dmamode    * auf FDC-Register umschalten
        jsr     rdstatus        * FDC-Status lesen
rdtend: move.w  varw3,sr        * Statusregister zurückholen
        rts                     * und zurück

rdtrerr2: bra   rdtend

rdtrerr1: bra   rdtend

*************************************************************************
*       Stellt den Lese-Kopf auf den in wtrack übergebenen Track        *
*************************************************************************

searcht: jsr    super           * Supervisor on
        jsr     track0          * Track null suchen
        move.w  #$86,dmamode    * Track Register selektieren
        move.w  wtrack,d6       * aktuellen Track an Trackregister
        jsr     wrcontr         * übergeben.
        move.w  #$80,dmamode    * FDC-Register selektieren
        move.w  #$1b,d6         * Search-Track-Befehl
        jsr     wrcontr         * an Controller übergeben
        move.l  #$60000,d7      * Timeout-Zähler
search1: subq.l #1,d7
        beq     searend1
        btst    #5,mfp          * Befehl schon abgearbeitet?
        bne     search1         * nein, dann weiter warten
        rts

searend1: move.w #-7,-(a7)      * Fehler = keine Diskette
        jsr     errhand
        rts
```

```
**********************************************************************
*  Track null suchen                                                 *
**********************************************************************

track0:  move.w   seek,d6         * Seek-Rate
         and.w    #3,d6           * mit Track null Befehl verknüpfen
         move.l   #$50000,d7      * Time-out-Zähler
         move.w   #$80,dmamode    * Zugriff auf FDC Register
         jsr      wrcontr         * Befehl übergeben

track0l1: subq.l  #1,d7           * Zähler erniedrigen
         beq      track0er        * Timeout
         btst     #5,mfp          * FDC fertig?
         bne      track0l1        * nein, dann weiter warten
         rts                      * und zurück

track0er: move.w  #-7,-(a7)       * Fehlernummer an Fehler-
         jsr      errhand         * handler übergeben
          rts                     * und zurück


**********************************************************************
*  übergibt die Adresse des Puffers platztr an den DMA-Controller    *
**********************************************************************

setplatz: move.l  #platztr,d0
         move.b   d0,dmalow
         lsr.l    #8,d0
         move.b   d0,dmamid
         lsr.l    #8,d0
         move.b   d0,dmahigh
         rts
```

```
************************************************************************
*  die Readtrack mit allen Syncs Steuerroutine, sie ruft alle er-     *
*  forderlichen Unterprogramme auf                                     *
************************************************************************


rdtracks: movem.l a3-a6/d3-d7,-(a7)    * Register retten
        jsr     cursmess
        jsr     delline
        jsr     cursmess
        move.l  #trfrag2,a0      * Message ausgeben
        jsr     printf
        move.w  #18,zeicount     * für Subroutine Dispbuf = 19 Zeilen
        jsr     super            * Supervisor on
        st      flock            * Floppy-interrupt aus
        jsr     seldrive         * Drive selektieren
        jsr     flreset          * Controller "resetten"
        jsr     searcht          * aktuellen Track suchen
        jsr     rdstrack         * zweimal diesen Track lesen, da die
        jsr     rdstrack         * Diskette sonst nicht auf Touren ist
        jsr     flreset          * FDC resetten
        jsr     user             * User-Mode on
        jsr     shtracks         * Diesen Track anzeigen
        jsr     super            * Supervisor on
        jsr     deselect         * Floppy deselektieren
        sf      flock            * Floppy-Interrupt freigeben
        jsr     user             * User-Mode einschalten
        movem.l (a7)+,a3-a6/d3-d7   * Register zurückholen
        rts                      * und zurück


************************************************************************
*  übergabe der Parameter zur Anzeige des Tracks an die allgemeine    *
*  showit-Routine                                                      *
************************************************************************

shtracks: move.w  #0,head2
        move.l  #platztr,topptr
```

```
        move.w  #18,zeicount         * 19 Zeilen auf Schirm
        move.w  #100,prcount         * 101 Zeilen auf den Drucken
        move.w  #7680,maxdown
        move.w  #7888,maxup
        jsr     cursbuf
        jsr     clrest               * Rest des Bildschirms löschen
        jsr     showit               * Puffer anzeigen, mit handle
        jsr     leerebuf             * der Cursortasten ect.
        rts

**********************************************************************
* lesen der Adressfelder auf der Diskette                            *
**********************************************************************


readadr: jsr    cursmess        * Cursor positionieren
        jsr     delline         * Linie löschen
        move.l  #hilcurs,a0     * Message ausgeben
        jsr     printf
        move.w  wdrive,d0
        cmp.w   #2,d0
        bgt     rdaderr
        jsr     super           * Supervisor on
        jsr     seldrive        * aktuelles Drive selektieren
        jsr     flreset         * FDC reset
        jsr     searcht         * zweimal den aktuellen Track suchen,
        jsr     searcht         * damit das Drive auf Touren kommt
        jsr     setplatz        * DMA-Transfer-Adresse setzen
readad1: jsr    rdadr           * Adressfelder lesen
        jsr     flreset         * FDC Reset
        jsr     user            * User-Mode on
        jsr     showadr         * Adressfelder anzeigen
        jsr     super           * Supervisor on
        jsr     deselect        * aktuelles Drive deselektieren
        jsr     user            * User-Mode on
        rts                     * und zurück

*********************************************************************
*                                                                   *
*  Liest 25 Adressfelder von der Diskette                           *
```

```
*                                                                       *
*************************************************************************


rdadr:   jsr      super            * Supervisor-Mode einschalten
         move.w   #$90,dmamode     * Togglen der read- write- Leitung
         move.w   #$190,dmamode    * Löschen des DMA-Status, Reset DMA
         move.w   #$90,dmamode     * Chips, auf Lesen und Sektorcount-
         move.w   #1,d6            * Register umschalten, 1 Sektor lesen
         jsr      wrcontr          * an FDC-Controller
         move.w   #$80,dmamode     * auf FDC-Register umschalten
         move.w   #24,d4           * 24+1 Adressfelder lesen
rdadr1:  move.w   #$c8,d6          * Read Adress Befehl
         move.l   #$40000,d7       * Time-out Zähler
         jsr      wrcontr          * Befehl an FDC
rdadr2:  btst     #5,mfp           * Befehl schon abgearbeitet?
         beq      rdadren1         * ja
         subq.l   #1,d7            * sonst Timeout verringern
         beq      rdaderr          * abgelaufen?, dann Fehler
         bra      rdadr2           * sonst weiterwarten
rdadren1: dbra    d4,rdadr1        * 25 mal wiederholen
         rts                       * und zurück

rdaderr: move.w   #-6,-(a7)        * Fehlermeldung
         jsr      errhand          * ausgeben, und abbrechen
         rts


*************************************************************************
* Anzeigen der eingelesenen Adressfelder, es wurden mehr Adressfelder *
* eingelesen, als angezeigt werden, da der DMA-Controller die Bytes    *
* in Gruppen von 16 überträgt, ein Adressfeld aber nur 6 Bytes be-     *
* inhaltet.                                                             *
*************************************************************************

showadr: jsr      cursmess         * Cursor pos. und löchen und Message
         jsr      delline          * ausgeben
         move.l   #sadfrag1,a0
         jsr      printf
         jsr      cursbuf          * cursor positionieren
```

```
        move.w   #17,d5          * 18 Adressfields anzeigen
        move.l   #platztr,a3     * Pufferadresse der Adressfelder
showadr1: move.w  #2,d4          * 3 Daten ausgeben (Track, Seite,
        move.w   #$20,-(a7)      * Sektor), erst ein Leerzeichen
        jsr      conout          * ausgeben
showadr2: move.b (a3)+,d0        * Byte aus Puffer holen
        move.w   d0,-(a7)        * als Wort auf den Stack schieben
        jsr      dezpr           * und als Dezimalzahl ausgeben
        move.w   #$20,-(a7)      * zwei Spaces hinterherschicken
        jsr      conout
        move.w   #$20,-(a7)
        jsr      conout
        dbra     d4,showadr2     * dreimal wiederholen
        move.w   #$20,-(a7)      * dann 2 Spaces schreiben
        jsr      conout
        move.w   #$20,-(a7)
        jsr      conout
        move.b   (a3)+,d0        * nächstes Byte aus Puffer (ent-
        ext.w    d0              * hält die Sektorgröße)
        move.w   #128,d1         * eine 0 bedeutet 128 Byte/Sektor
        cmp.w    #0,d0
        beq      showadr7
        move.w   #256,d1
        cmp.w    #1,d0           * 1 gleich 256 Byte/Sektor
        beq      showadr7
        move.w   #512,d1
        cmp.w    #2,d0           * 2 gleich 512 Byte/Sektor
        beq      showadr7
        move.w   #1024,d1        * sonst 1024 Byte/Sektor als Default
showadr7: move.w  d1,-(a7)       * Anzahl der Byte/Sektor als
        jsr      dezpr           * Dezimalzahl ausgeben
        move.w   #$20,-(a7)      * ein Space und
        jsr      conout
        move.l   #spaces,a0      * mehrere Spaces ausgeben
        jsr      printf
        move.b   (a3)+,d0        * nächstes Byte aus Puffer ist die
        move.w   d0,-(a7)        * Checksumme des Adressfeldes, welche
        jsr      hexpr          * nun als Sedezimalzahl ausgegeben wird
        move.b   (a3)+,d0        * nächstes Byte aus dem Puffer
```

```
        move.w  d0,-(a7)
        jsr     hexpr           * als Sedezimalzahl

        move.w  #13,-(a7)       * Carriage-Return plus Linefeed
        jsr     conout          * hinterherschicken
        move.w  #10,-(a7)
        jsr     conout
        dbra    d5,showadr1     * 18 mal wiederholen
        jsr     wtast           * und auf Tastendruck warten
        rts                     * nun zurück
```

Nun die Unterprogramme des CLUSTER-Menues

```
************************************************************************
************************************************************************
* Subroutines des Menuepunktes CLUSTER des Hauptmenues, die Routinen *
* greifen auf Routinen des OPTION-Menues zurück, daher bitte         *
* das OPTION-Menue zuerst implementieren                             *
************************************************************************
************************************************************************

edclust: jsr    cursmess        * Cursor pos. ect.
         jsr    delline
         move.w #20,spalte
         move.w #2,zeile
         jsr    loccurs
         move.l #edfrag1,a0     * Message
         jsr    printf
         move.w #512,maxdown    * Scroll up and down Variablen
         move.w #720,maxup
         move.l #platztr,editptr  * Pufferadresse
         jsr    editit          * Cluster editieren
         jsr    cursmess        * Message-Zeile löschen
         jsr    delline
         jsr    cursmess
         move.l #clfrag1,a0     * Message ausgeben
         jsr    printf
```

```
        jsr     curlinks        * drei mal links
        jsr     curlinks        * zum Sprung ins read
        jsr     curlinks        * Untermenue
        jsr     shclust         * Anzeigen des Clusters
        rts                     * und zurück


**************************************************************************
*  erniedrigen der Clusternummer im Clustermenue                         *
**************************************************************************


decclust: move.w  #-1,-(a7)
        move.w  #11,-(a7)       * Keyboard-Shift abfragen
        trap    #13             * wenn eine Shift-Taste betätigt
        addq.l  #4,a7           * wurde, dann beträgt das Decrement
        btst    #0,d0           * 10, sonst wird nur um eins er-
        bne     decclshi        * niedrigt
        btst    #1,d0
        bne     decclshi        * Shift betätigt
        move.w  #1,d2           * sonst nicht Shift betätigt und
        bra     declst0         * daher nur um 1 erniedrigen
decclshi: move.w #10,d2         * um 10 erniedrigen
declst0:  move.w wclust,d0      * aktuelle Clusternummer
        sub.w   d2,d0           * Decrement subtrahieren
        cmp.w   #0,d0           * 0 schon überschritten
        blt     declst1         * ja
        bra     declst2         * nein
declst1: move.w  maxclust,d0    * maximale Clusternummer als neue akt.
declst2: move.w  d0,wclust      * Clusternummer speichern
        ext.l   d0              * nun muß die neue aktuelle Cluster-
        divu    #1000,d0        * nummer auch in das Menue eingetragen
        add.b   #'0',d0         * werden, daher durch dividieren in
        move.b  d0,m1clusa1     * Zehnerpotenzen zerlegen, und ins
        swap    d0              * Clustermenue eintragen
        ext.l   d0
        divu    #100,d0
        add.b   #'0',d0
        move.b  d0,m1clusa1+1   * 100'er eintragen
        swap    d0
        ext.l   d0
```

```
        divu     #10,d0
        add.b    #'0',d0
        move.b   d0,m1clusa1+2  * 10'er eintragen
        swap     d0
        add.b    #'0',d0
        move.b   d0,m1clusa1+3  * und zum Schluß die Einer ins Menue
        jsr      dispmen        * Menue anzeigen und
        rts                     * zurück


*************************************************************************
* erhöhen der aktuellen Clusternummer                                   *
*************************************************************************

incclust: move.w #-1,-(a7)
        move.w   #11,-(a7)      * Keyboard-Shift abfragen
        trap     #13            * wie bei decclust
        addq.l   #4,a7
        btst     #0,d0
        bne      incclshi
        btst     #1,d0
        bne      incclshi       *
        move.w   #1,d2          * keine Shift-Taste, dann 1 als Incre-
        bra      inclst0        * ment
incclshi: move.w #10,d2         * sonst Increment gleich 10
inclst0: move.w wclust,d0       * Increment zu aktueller Clusternummer
        add.w    d2,d0          * addieren, und mit maximaler Nummer
        cmp.w    maxclust,d0    * vergleichen
        blt      inclst1        * kleiner als maximale Nummer
        move.w   #0,d0          * Nummer 0 annehmen
inclst1: move.w  d0,wclust      * neue aktuelle Nummer speichern
        ext.l    d0             * und die aktuelle Nummer ins Menue-
        divu     #1000,d0       * eintragen, 1000'er Stelle
        add.b    #'0',d0
        move.b   d0,m1clusa1    * 1000'er Stelle eintragen
        swap     d0
        ext.l    d0
        divu     #100,d0
        add.b    #'0',d0
```

```
        move.b   d0,m1clusa1+1   * 100'er Stelle eintragen
        swap     d0
        ext.l    d0
        divu     #10,d0
        add.b    #'0',d0
        move.b   d0,m1clusa1+2   *10'er Stelle eintragen
        swap     d0
        add.b    #'0',d0
        move.b   d0,m1clusa1+3   * 1'er Stelle eintragen
        jsr      dispmen         * Menue anzeigen und
        rts                      * zurück

***********************************************************************
*  sucht den nächsten, auf den aktuellen Cluster folgenden Cluster,  *
*  ist kein Nachfolge-cluster vorhanden, wird dies kundgetan          *
***********************************************************************


nextclst: move.w wclust,d0       * aktuelle Clusternummer
        move.w   d0,oldclst      * zwischenspeichern
        jsr      findclst        * nächsten Cluster suchen
        move.w   newclst,d0      * hier steht der nächste Cluster
        tst.w    d0              * oder eine 0, was einen Fehler
        beq      neclerr1        * signalisiert
        cmp.w    #$ff8,d0        * oder ein Ende-kennzeichen, was
        bge      neclerr1        * den letzten Cluster anzeigt.
        subq.w   #1,d0           * eins subtrahieren, zum besseren
        move.w   d0,wclust       * Handling der Menueanzeige, nun
        move.l   #3,revnum       * kann nämlich die incclust-Routine
        jsr      incclust        * aufgerufen werden, die den incre-
        jsr      rdclust         * mentierten Cluster anzeigt, dann
neclend: rts                     * diesen Cluster lesen

neclerr1: jsr    cursmess
        move.w   #-19,-(a7)      * Letzten Cluster als solchen melden
        jsr      errhand
        jsr      cursmess
```

```
        move.l   #clfrag1,a0
        jsr      printf
        bra      neclend           * und zurück


*************************************************************************


findclst: move.l #fatbuf,a0        * Adresse des FAT-Buffers
        move.w   oldclst,d0        * alte Clusternummer
        move.w   #3,d1             * mal 3, und
        mulu     d0,d1
        lsr.w    #1,d1             * dividiert durch 2, gleich mal 1.5
        btst     #0,d0             * war die alte Clusternr. gerade oder
        bne      ungerad           * ungerade (durch 2 teilbar odernicht)
gerade: move.b   1(a0,d1.w),d0     * wenn gerade, dann mostsignificant
        lsl.w    #8,d0             * Nibble holen, 8 Bitstellen nach
        or.b     0(a0,d1.w),d0     * links schieben, und die beiden rest-
        and.w    #$0fff,d0         * lichen Nibble dazuodern
        move.w   d0,newclst        * als neue Clusternummer speichern
        bra      ficlend           * und zurück
ungerad: move.b  1(a0,d1.w),d0     * sonst: most significant Nibble und
        lsl.w    #8,d0             * nächstes Nibble holen, 8 Bitstellen
        move.b   0(a0,d1.w),d0     * nach links, least significant Nibble
        lsr.w    #4,d0             * die oberen 12-Bit enthalten die
        and.w    #$0fff,d0         * Clusternummer, daher 4 Bit nach
        move.w   d0,newclst        * nach rechts, ausmaskieren, speichern
ficlend: rts                       * und zurück


*************************************************************************
*  schreibt den aktuellen Cluster nach Frage auf die Diskette           *
*************************************************************************


wrclust: movem.l  a3-a5/d3-d5,-(a7) * Register retten
        move.w   #0,spalte
        move.w   #2,zeile          * Message ausgeben
        jsr      loccurs
        move.l   #clfrag5,a0
```

```
        jsr     printf
        move.l  #m1secta,a3     * aktuellen Track
        move.w  #9,d3
wrcl1:  move.b  (a3)+,d0
        move.w  d0,-(a7)
        jsr     conout
        dbra    d3,wrcl1
        move.l  #m1clusa,a3     * und Cluster als Frage ausgeben
        move.w  #12,d3
wrcl2:  move.b  (a3)+,d0
        move.w  d0,-(a7)
        jsr     conout
        dbra    d3,wrcl2
        move.l  #wrfrag2,a0
        jsr     printf
        jsr     leerebuf
        jsr     wtast           * wirklich schreiben
        cmp.b   #'y',d0         * ja
        beq     writclst
        cmp.b   #'Y',d0         * ja
        bne     wrclend1        * sonst halt nicht schreiben
writclst: move.w wdrive,-(a7)   * aktuelles Drive übergeben
        move.w  wclust,d0       * aktuelle Clusternummer
        sub.w   #2,d0           * Nummer 2 ist erster Datencl.
        muls    clsiz,d0        * logische Sektornummer
        add.w   datrec,d0       * berechnen
        move.w  d0,-(a7)        * logische Sektornummer auf ST
        move.w  clsiz,-(a7)     * Anzahl Sektoren pro Cluster
        move.l  #platztr,-(a7)  * Anfangsadresse des Clusters
        move.w  #3,-(a7)        * Schreiben, Diskettenwechsel ignor.
        move.w  #4,-(a7)        * Rwabs
        trap    #13             * BIOS-Trap
        add.l   #14,a7          * Stack restaurieren
        tst.w   d0              * Fehler aufgetreten,
        bmi     wrclster        * wenn ja, dann handlen
wrclend: jsr    cursmess
        jsr     delline         * sonst Menue-Message ausgeben
        jsr     cursmess
        move.l  #clfrag1,a0
```

```
        jsr       printf
        movem.l   (a7)+,a3-a5/d3-d5  * Register zurückholen
        rts                  * und zurück


wrclster: move.w d0,-(a7)        * Fehlernummer an
        jsr       errhand        * Errorhandler und anzeigen
        bra       wrclend        * und zurück

wrclend1: jsr     cursmess
        jsr       delline
        jsr       cursmess
        move.l    #wrfrag3,a0
        jsr       printf
        jsr       leerebuf
        jsr       wtast
        jsr       delline
        bra       wrclend
        rts


*************************************************************************
*  liest den aktuellen Cluster in den Speicher, funktioniert auch      *
*  mit der RAM-Disk                                                     *
*************************************************************************


rdclust: movem.l  a3-a6/d3-d7,-(a7)     * Register retten
rdcl0:   move.w   wdrive,-(a7)     * aktuelles Drive
         move.w   wclust,d0        * aktueller Cluster
         subq.w   #2,d0            * logische Sektornummer berechnenen
         muls     clsiz,d0
         add.w    datrec,d0
         tst.w    d0
         bpl      rdcl2            * größer als null
         move.w   #0,d0            * wenn nicht, dann null annehmen
rdcl2:   move.w   d0,logsect       * logischen Sektor speichern
         move.w   d0,-(a7)         * und auf Stack
         move.w   #2,-(a7)         * 2 Sektoren lesen
         move.l   #platztr,-(a7) * Pufferadresse
```

```
        move.w   #0,-(a7)         *
        move.w   #4,-(a7)         * Rwabs Befehl
        trap     #13              * BIOS-Trap
        add.l    #14,a7           * Stack restaurieren
        tst.w    d0               * ist ein Fehler aufgetreten?
        bmi      rdclster         * wenn ja, dann anzeigen
        move.w   logsect,d0       * logischen Sektor in physikalisch.
        divs     #9,d0            * hier muß noch geändert werden
        swap     d0               * umrechnen
        addq.w    #1,d0           * eins zum Rest der Div. addieren
        move.w   d0,wsector       * gleich physiklalischer Sektor
        swap     d0
        move.w   d0,d2            * Ergebnis der Div. speichern
        move.w   #0,wside         * Seite null als Default
        move.w   anzside,d1       * Anzahl der Seiten
        cmp.w    #2,d1            * wenn 2 Seiten,
        bne      rdcl3
        lsr.w    #1,d0            * dann durch 2 teilen
        btst     #0,d2            * Test ob Ergebnis ungerade
        beq      rdcl4            *
        move.w   #1,wside
rdcl3:
rdcl4:  move.w   d0,wtrack        * gleich physikalischer Sektor
        jsr      secinmem         * Sector into Memory
        jsr      shclust          * Cluster anzeigen
rdclend: jsr     cursmess
        move.l   #clfrag1,a0      * Message ausgeben
        jsr      printf
        movem.l  (a7)+,a3-a6/d3-d7  * Register zurückholen
        rts                       * und zurück

rdclster: jsr    initdriv         * wemm Fehler aufgetreten ist, dann
        tst.l    d0               * erst Drive initialisieren,
        bne      rdcl0            * wenn dies ohne Fehler, dann nochein.
        move.w   d0,-(a7)
        jsr      errhand          * muß noch geändert werden
        bra      rdclend
```

```
secinmem: move.w  wside,d0        * überträgt den aktuellen Sektor ins
         add.b   #'0',d0         * Sektormenu zwecks späterer Anzeige
         move.b  d0,mside
         move.w  wsector,d0
         ext.l   d0
         divs    #10,d0
         add.b   #'0',d0
         move.b  d0,msector
         swap    d0
         add.b   #'0',d0
         move.b  d0,msector+1   * low Byte des Sektors
         move.w  wtrack,d0
         ext.l   d0
         divs    #10,d0
         add.b   #'0',d0
         move.b  d0,mtrack
         swap    d0
         add.b   #'0',d0
         move.b  d0,mtrack+1     * low Byte des Track
         rts


***************************************************************************
*  Zeigt den Cluster an                                                   *
***************************************************************************


shclust: move.w  #0,head2        * Byte-Zähler
         move.w  #18,zeicount    * 19 Zeilen sollen angezeigt werden
         move.w  #63,prcount     * bei Druckerausgabe 64 Zeilen
         move.l  #platztr,topptr  * Pufferadresse
         move.w  #512,maxdown    * Scrollbegrenzer
         move.w  #720,maxup
         jsr     cursbuf         * Cursor positionieren und
         jsr     clrest          * Rest des Bildschirms löschen
         move.w  #0,spalte       * Cursor auf letzte Bildschirmzeile
         move.w  #24,zeile
         jsr     loccurs         * positionieren
         move.l  #clfrag2,a0
```

```
         jsr      printf
         jsr      showit          * und den Cluster anzeigen
         rts                      * anschließend zurück


*****************************************************************
*  Anzeigen des Startcluster der auf der Diskette befindlichen Files  *
*  die Startcluster werden durch <return> in die aktuelle Menue-      *
*  Clusternummer übernommen, ist der invers hervorgehobene Filename   *
*  ein Subdirectory, wird in dieses verzweigt.                        *
*****************************************************************


stclust: movem.l  a3-a5/d3-d7,-(a7)
         jsr      initdriv        * Drive intialisieren
         jsr      rdfat           * FAT- und Directory-Sektoren in die
         jsr      rddir           * jeweiligen Puffer einlesen
stclst0:   move.w   #0,spalte
         move.w   #2,zeile
         jsr      loccurs         * Cusor positionieren
         move.l   #sclfrag1,a0
         jsr      printf
         move.w   #17,zeicount    * 18 zeilen sollen angezeigt werden
         jsr      delrest         * Rest der Zeile löschen
         move.l   #dirbuf,a3      * Adresse des Directory-Buffers
         move.l   a3,a4           * zwischenspeichern
         move.l   a3,topptr       * als Zeiger benutzen
         move.l   a3,oldtop       * nochmal zwischenspeichern
         jsr      cursbuf         * Cursor
         jsr      showdir         * 18 Zeilen anzeigen
         jsr      cursbuf         * Cursor auf Anfang
         move.l   #dirbuf,topptr  * Anfang des Directory-Puffers
         jsr      revon           * Invers einschalten
         jsr      dirzeil         * ersten Filenamen (Diskettenname)
         jsr      revout          * invers überschreiben, dann invers
stclst1: jsr      taste           * wieder aus, und Tastatur abfragen
         swap     d0
         cmp.b    #$1c,d0         * Return Taste betätigt?
         beq      dirclsel        * wenn ja
```

```
         cmp.b    #$48,d0        * cursor up?
         beq      stclup         * ja
         cmp.b    #$50,d0        * Cursor down?
         beq      stcldo         * ja
         cmp.b    #$4b,d0        * Cursor links?
         beq      stclli         * ja
         cmp.b    #$4d,d0        * Cursor rechts
         bne      stclst1
         jsr      currecht       * ja, dann aufrufen
         bra      stclend1
stclli:  jsr      curlinks
         bra      stclend1


stclup:  move.w   zeile,d0       * aktuelle Cursorzeile
         cmp.w    #4,d0          * Zeile 4 gleich obere Grenze
         ble      stclup3        * gleich 4, dann scrollen
         move.w   #0,spalte
         jsr      loccurs
         jsr      dirzeil        * sonst eins von der aktuellen
         subq.w   #1,zeile       * Zeile subtrahieren, Cursor
         move.w   #0,spalte      * auf die neue Zeile und Spalte
         jsr      loccurs        * null einstellen
         jsr      revon
         jsr      dirzeil        * und diese Zeile invers ausgeben
         jsr      revout         * invers ausschalten
         bra      stclupen       * und zurück

stclup3: cmp.l    #dirbuf,topptr * oberste Zeile im Puffer erreicht?
         beq      stclupen       * wenn ja, nicht scrollen son. back
         move.l   topptr,d0      * sonst Zeiger um Anzahl der Zeilen
         move.w   zeicount,d0    * mal Anzahl der Zeichen pro Zeile
         addq.w   #1,d0          * erniedrigen geändert 18.8.86
         muls     #32,d0
         sub.l    d0,topptr      * Zeiger in Buffer erniedrigen
         jsr      showdir        * 18 Zeilen anzeigen
         move.w   #21,zeile
         move.w   #0,spalte      * letzte angezeigte Zeile invers
         jsr      loccurs        * Cursor pos.
         jsr      revon          * invers on
```

```
        jsr     dirzeil         * Zeile anzeigen
        jsr     revout          * invers wieder aus
        jsr     leerebuf
stclupen: bra    stclst1        * und zum Loop


stcldo: move.w  zeile,d0        * aktuelle Zeile größer als 20
        cmp.w   #20,d0
        bgt     stcldo3         * ja
        move.w  zeile,d0        * wenn nicht, dann eins addieren
        addq.w  #1,d0           *
        sub.w   #4,d0           * Offset zum oberen Bildschirmrand
        ext.l   d0
        lsl.l   #5,d0           * mit 32 multiplizieren
        move.l  topptr,a6       * Zeiger in Directorypuffer
        move.b  0(a6,d0.l),d0   * erstes Byte dieses Eintrages holen
        beq     stcldoen        * wenn Byte=0, dann leerer Eintrage
        move.w  #0,spalte       * sonst Cursor positionieren und
        jsr     loccurs
        jsr     dirzeil         * die alte Zeile normal anzeigen
        addq.w  #1,zeile        * Zeilenzähler erhöhen und
        move.w  #0,spalte
        jsr     loccurs
        jsr     revon           * die neue Zeile invers
        jsr     dirzeil         * darstellen
        jsr     revout          * inverse Darstellung wieder aus
        jsr     loccurs
stcldo1: bra    stcldoen        * und zum Loop

stcldo3: move.w zeile,d0        * erstes Byte des nächsten Directory-
        addq.w  #1,d0           * eintrages holen, eins addieren und
        sub.w   #4,d0           * Offset zum oberen Rand subtrahieren
        ext.l   d0
        lsl.l   #5,d0           * mal 32 (Anzahl Byte pro Direintrag)
        move.l  topptr,a6       * Zeiger auf Anfang Dirpuffer
        move.b  0(a6,d0.l),d0   * ist der nächste Eintrag null
        beq     stcldoen        * dann zurück zum Loop
        move.w  zeicount,d0     * Anzahl der anzuzeigenden Zeilen
        addq.w  #1,d0           * plus eins
        muls    #32,d0          * mal 32 gleich Offset vomPufferanfang
```

```
        add.l    topptr,d0       * Offset zu topptr addieren
        move.l   d0,topptr
        jsr      cursbuf         * Cursor pos.
        jsr      showdir         * Directory anzeigen
        jsr      cursbuf         * und den ersten Eintrag invers
        jsr      revon
        jsr      dirzeil         * darstellen
        jsr      revout
        jsr      leerebuf
stcldoen: bra     stclst1        * zurück zum Loop

stclend1: jsr    cursmess        * Messageline löschen
        jsr      delline
        jsr      cursmess
        move.l   #clfrag1,a0     * Mode anzeigen
        jsr      printf
        movem.l  (a7)+,a3-a5/d3-d7  * Register zurückholen
        rts                      * und zurück


*************************************************************************
* reagiert auf Betätigung der RETURN Taste, übernimmt den               *
* oder zeigt ein Subdirectory an                                        *
*************************************************************************

dirclsel: move.l  topptr,a0      * Zeiger auf momentanen Pufferanfang
        move.w   zeile,d0
        sub.w    #4,d0           *
        ext.l    d0
        lsl.l    #5,d0           * mal 32
        move.b   11(a0,d0.l),d1  * File-Type Byte holen
        cmp.b    #$10,d1         * ist es ein Subdirectory
        beq      subdir          * ja

dirsel1: move.w   clstnum,d0     * sonst aktuelle Clusternummer
        subq.w   #1,d0           * 1 subtrahieren wegen incclust
        move.w   d0,wclust
```

```
        move.l  #3,revnum       * 3. Menuepunkt invers
        jsr     incclust        * Clusternummer erhöhen und Menue an-
        bra     stclend1        * zeigen, dann zum Loop

************************************************

dirsel2: jsr    rddir           * Directory-Sektoren neu einlesen
        bra     subdiren

subdir: tst.w   clstnum         * ist die Clusternummer null dann nur
        beq     dirsel2         * die Directory-Sektoren neu einlesen
        move.w  clstnum,d0      * sonst ab der Anfangsclusternummerdes
        move.l  #dirbuf,dirptr  * Subdirectories 2 logische Skt. lesen
        clr.w   d3
subdir1: move.w clstnum,d0
        move.w  wdrive,-(a7)    * aktuelles Drive
        subq.w  #2,d0           * Clusternummer in logische Sektornum.
        muls    clsiz,d0        * umrechnen
        add.w   datrec,d0
        move.w  d0,-(a7)
        move.w  #2,-(a7)        * 2 logische Sektoren lesen
        move.l  dirptr,-(a7)    * Pufferadresse
        move.w  #2,-(a7)        * unbedingt lesen
        move.w  #4,-(a7)        * BIOS Rwabs
        trap    #13             * BIOS Trap
        add.l   #14,a7
        tst.w   d0              * Fehler aufgetreten?
        bmi     subdierr        * ja, dann handlen
        add.l   #1024,dirptr    * sonst 1024 Byte pro Cluster add.
        move.w  clstnum,oldclst  * Clusternummer speichern
        jsr     findclst        * und evt. Nachfolgecluster suchen
        move.w  newclst,d0
        move.w  d0,clstnum
        tst.w   d0              * Nachfolgecluster gefunden?
        beq     subdiren        * wenn nicht, dann zum Ende
        cmp.w   #$ff8,d0        * war es ein Endekennzeichen
        bge     subdiren        * wenn ja, dann zum Ende
        bra     subdir1         * sonst zweiter einlesen
```

```
subdiren: bra      stclst0         * zum Loop

subdierr: move.w   d0,-(a7)        * Fehler handlen, Fehlernummer
          jsr      errhand
          bra      stclend1

*************************************************************************
* zeigt eine Seite Directoryeinträge                                    *
*************************************************************************

showdir: move.w    #0,eflag
         jsr       cursbuf         * Cursor pos.
         jsr       clrest          * Rest des Bildschirms löschen
         move.l    topptr,a5       * Zeiger in Dirpuffer
         move.w    zeicount,d7     * Anzahl der Zeilen pro Seite
showd1:  move.b    #' ',d0         * erst mal Spaces ausgeben
         move.w    d0,-(a7)
         jsr       conout
         move.b    #' ',d0
         move.w    d0,-(a7)
         jsr       conout
         clr.l     d4
         move.w    #9,d6           * Länge von Filnamen mit EXT.
         move.b    0(a5,d4.l),d0    * Test ob leerer Dir-Eintrag
         beq       showdien        * wenn ja, dann zum Ende
         addq.l    #1,d4           * wenn nicht, dann
         move.w    d0,-(a7)
         jsr       conout
showd2:  move.b    0(a5,d4.l),d0   * den Filenamen mit EXT. ausgeben
         addq.l    #1,d4
         move.w    d0,-(a7)
         jsr       conout
         dbra      d6,showd2
         move.w    #20,tab1
         jsr       curstab         * nun das Fileattribut
         jsr       disattr         * ausgeben
         move.w    #40,tab1
         jsr       curstab
         jsr       disclus         * anschließend den Startcluster aus.
```

```
        move.w  #55,tab1
        jsr     curstab
        jsr     dissize         * und schließlich die Filegröße in B.
        move.w  #0,spalte
        addq.w  #1,zeile
        jsr     loccurs
        add.l   #32,a5          * 32 Byte pro Dir-Eintrag
        dbra    d7,showd1
showd8: move.w  #0,spalte       * Übernahmemessage in letzter Zeile
        move.w  #24,zeile       * ausgeben und
        jsr     loccurs
        move.l  #sclfrag2,a0
        jsr     printf
        rts                     * zurück

showdien: move.w #1,eflag
        bra     showd8


*********************************************************************
* gibt eine Directory Zeile auf dem Bildschirm aus                  *
*********************************************************************

dirzeil: move.l topptr,a3       * Zeiger in Dir-Puffer
        move.w  #0,eflag
        move.w  zeile,d3
        sub.w   #4,d3           * Offset vom oberen Bildschirmrand
        ext.l   d3
        lsl.l   #5,d3           * mal 32, entspricht einem Dir-Eintrag
        move.b  #' ',d4
        move.w  d4,-(a7)        * zwei Spaces
        jsr     conout
        move.w  d4,-(a7)
        jsr     conout
        move.b  0(a3,d3.l),d0     * erstes Byte des Eintrages, wenn
        beq     dirzend1        * null, dann leerer Eintrag
        move.w  d0,-(a7)        * sonst Byte ausgeben
        jsr     conout
        addq.l  #1,d3
        move.w  #9,d6           * restliche Länge des Eintrages
```

```
dirzei1: move.b  0(a3,d3.l),d0  * restlichen Byte des Eintrages holen
         addq.l  #1,d3
         move.w  d0,-(a7)
         jsr     conout         * und ausgeben
         dbra    d6,dirzei1
         move.w  #20,tab1
         jsr     curstab
         jsr     disattr        * nun das Fileattribut ausgeben
         move.w  #40,tab1
         jsr     curstab
         jsr     disclus        * und den Startcluster
         move.w  #55,tab1
         jsr     curstab
         jsr     dissize        * schließlich die Filegröße in Byte
dirzend: rts                    * und zurück

dirzend1: move.w #1,eflag
         bra     dirzend

**************************************************************************
*  gibt die Startclusternummer des aktuellen Dir-Eintrages aus           *
**************************************************************************

disclus: move.b  #' ',d0
         move.w  d0,-(a7)
         jsr     conout         * Space ausgeben
         move.w  zeile,d0
         sub.w   #4,d0
         ext.l   d0
         lsl.l   #5,d0          * mal 32
         move.b  27(a3,d0.l),d1 * auf Startcluster im Eintragzugreifen
         lsl.w   #8,d1          * mal 256 da High-Byte
         move.b  26(a3,d0.l),d1 * Low-Byte dazuladen
         move.w  d1,clstnum     * als Clusternummer speichern
         move.w  d1,-(a7)
         jsr     dezpr          * und als Dezimalzahl ausgeben
         move.b  #' ',d0        * noch ein Space und
         move.w  d0,-(a7)
         jsr     conout
         rts                    * zurück
```

```
***********************************************************************
*  gibt die Filegröße in Byte des aktuellen Dir-Eintrages aus         *
***********************************************************************

dissize: move.w  zeile,d3
         sub.w   #4,d3            * Offset vom oberen Bildschirmrand
         ext.l   d3               * subtrahieren
         lsl.l   #5,d3            * mal 32
         clr.l   d1
         move.l  topptr,a3
         move.b  31(a3,d3.l),d1   * Most significant Byte zuerst
         lsl.l   #8,d1            * (Intel), Byte in nächste Byte-Pos.
         move.b  30(a3,d3.l),d1   * schieben, next signif. Byte dazu-
         lsl.l   #8,d1            * laden und wiederum um eine Byte-
         move.b  29(a3,d3.l),d1   * Position nach links schieben
         lsl.l   #8,d1            * bis alle vier Byte des Größen-
         move.b  28(a3,d3.l),d1   * eintrages erfaßt und ins
         move.l  d1,-(a7)         * Motorola-Format umgewandelt wurd.
         jsr     dezlpr           * dann die Größe als Dez.-Zahl aus-
         move.w  #$20,-(a7)       * geben, noch ein Space hinterher
         jsr     conout
         rts                      * und zurück


***********************************************************************
*  gibt das File-Attribut des aktuellen Directory-Eintrages aus       *
***********************************************************************

disattr: move.w  zeile,d3         * aktuelle Zeile
         sub.w   #4,d3            * Offset zum oberen Bildschirmrand
         ext.l   d3
         lsl.l   #5,d3            * mal 32
         move.l  topptr,a3        * Zeiger in Dir.-Puffer
         move.b  0(a3,d3.l),d1    * erstes Byte des Eintrages holen
         cmp.b   #$e5,d1          * ist es das Zeichen für gelöschtes
         beq     deleted          * File? wenn ja, dann anzeigen
         move.b  11(a3,d3.l),d1   * sonst Dateiattribut holen,
         cmp.b   #$10,d1          * ist es ein Subdirectory? wenn ja,
```

```
          beq       folder          * dann 'Subdirectory' ausgeben
          cmp.b     #$01,d1         * ist es nur zum Lesen zu öffnen
          beq       readonly
          cmp.b     #$02,d1         * handelt es sich um ein verstecktes
          beq       hidden          * File?
          cmp.b     #$08,d1         * ist es der Diskettenname
          beq       disname
          move.l    #treadwr,a0     * Default ist: File kann gelesen und
          jsr       printf          * beschriebeben werden
disatten: rts                       * und zurück

disname:  move.l    #tdisname,a0    * ausgeben und
          jsr       printf
          bra       disatten        * zurück

folder:   move.l    #tfolder,a0     * ausgeben und
          jsr       printf
          bra       disatten        * zurück

readonly: move.l    #treadon,a0     * ausgeben und
          jsr       printf
          bra       disatten        * zurück


hidden:   move.l    #thidden,a0     * ausgeben und
          jsr       printf
          bra       disatten        * zurück

deleted:  move.l    #tdelet,a0      * ausgeben und
          jsr       printf
          bra       disatten        * zurück
```

Und schließlich die Unterprogramme des FORMAT-Menues inclusive GAP-Menue

```
*************************************************************************
*************************************************************************
*   Format Unterroutinen des Hauptmenues                                *
*   diese Routinen greifen auf Unterprogramme des Menuepunktes          *
*   TRACK with SYNCS zurück, daher muß der Menuepunkt TRACK with        *
*   SYNCS zuerst implementiert werden, und danach erst der Formatter   *
*************************************************************************
*************************************************************************

format1: jsr      cursmess      * Cursor in Messageline positionieren
         jsr      delline       * Zeile löschen
         jsr      leerebuf      * und Tastaturbuffer löschen
         move.l   #fofrag2,a0   * Fragen ob wirklich formattieren
         jsr      printf
         move.w   wtrack,-(a7)
         jsr      dezpr
         move.l   #fofrag5,a0
         jsr      printf
         move.w   wside,-(a7)
         jsr      dezpr
         move.l   #fofrag6,a0
         jsr      printf
         move.w   wdrive,-(a7)
         jsr      dezpr
         move.l   #fofrag3,a0
         jsr      printf
         jsr      wtast         * Tastatur abfragen
         cmp.b    #'y',d0
         beq      doform1
         cmp.b    #'Y',d0
         beq      doform1
         jsr      delline        * wenn weder großes noch kleines 'y',
         jsr      leerebuf       * dann nicht formattieren sondern
         move.l   #fofrag4,a0
         jsr      printf
         jsr      wtast          * auf Tastendruck warten und
         bra      form1end       * zurück

doform1:  move.w   #$e5e5,-(a7)      * sonst formattieren, Virgin wert
         move.l   #$87654321,-(a7)  * Magic number
```

```
        move.w  #1,-(a7)        * sector-interleave
        move.w  wside,-(a7)     * aktuelle Seite
        move.w  wtrack,-(a7)    * aktueller Track
        move.w  asector,-(a7)   * aktuelle Anzahl derSektoren/Track
        move.w  wdrive,-(a7)    * aktuelles Drive
        clr.l   -(a7)           * Dummy Langwort
        move.l  #formbuf,-(a7)  * Platz zur Erzeugung des Tracks
        move.w  #10,-(a7)       * XBIOS Flopfmt
        trap    #14             * XBIOS Trap
        add.l   #26,a7
        tst.w   d0
        bmi     form1err        * Fehler aufgetreten?
form1end: jsr    cursmess
        jsr     delline
        jsr     cursmess
        move.l  #fofrag1,a0     * Message ausgeben
        jsr     printf
        jsr     currecht        * menue-Cursor korrigieren, Menue
        rts                     * anzeigen und zurück

form1err: move.w d0,-(a7)       * Fehlernummer auf Stack, handlen
        jsr     errhand
        bra     form1end        * und zurück


***********************************************************************
*   erzeugt ab der Adresse formbuf einen Track mit allen syncs, der   *
*   dann mit writetr des Controller zwecks Formatierung auf Diskette  *
*   geschrieben wird.  xformat im Menue                               *
***********************************************************************


maketr: move.w  #1,sektor      * erster Sektor hat die Nummer 1
        move.l  #formbuf,a2    * Adresse des Puffers, in dem der Tr.
        move.w  gap1,d0        * erzeugt wird, erste Lücke
        move.w  #$4e,d7        * Vorspann-Byte ist $4E
        jsr     wpuff          * gap1 mal in Puffer eintragen
makt1:  move.w  gap2,d0        * Anzahl Byte zweite Lücke
        move.w  #0,d7          * Byte-Wert ist 0
        jsr     wpuff          * in Puffer eintragen
```

```
        move.w  #3,d0       * drei mal $F5 in Puffer als Syncbyte
        move.b  #$f5,d7     * und zum löschen des CRC-Registers
        jsr     wpuff       * (Checksum), in Puffer eintragen
        move.b  #$fe,(a2)+  * Adressmark direkt in Puffer
        move.w  wtrack,d0
        move.b  d0,(a2)+    * ebenso den aktuellen Track
        move.w  wside,d0
        move.b  d0,(a2)+    * die aktuelle Seite
        move.w  sektor,d0
        move.b  d0,(a2)+    * den momentanen Sektor
        move.w  drbyte,d0   * Anzahl der Byte pro Sektor
        cmp.w   #1024,d0
        beq     makt2
        cmp.w   #512,d0     * mit den 4 möglichen Werten
        beq     makt3       * vergleichen, und demnach den
        cmp.w   #256,d0
        beq     makt4
        move.w  #0,d1
        bra     makt5
makt4:  move.w  #1,d1
        bra     makt5
makt3:  move.w  #2,d1
        bra     makt5
makt2:  move.w  #3,d1
makt5:  move.b  d1,(a2)+    * geforderten Wert eintragen
        move.b  #$f7,(a2)+  *  Checksum of Adressfield in Puff.
        move.w  gap3,d0     * Anzahl der Byte in Lücke 3
        move.w  #$4e,d7     * Füllbyte ist $4E
        jsr     wpuff       * in Puffer schreiben
        move.w  gap2,d0     * und nocheinmal gap2 mal 0 in
        move.w  #0,d7       * den Puffer eintragen
        jsr     wpuff
        move.w  #3,d0       * 3 mal Syncbbyt, werden als A1
        move.w  #$f5,d7     * auf Diskette geschrieben, als
        jsr     wpuff       * $F5 in Puffer schreiben
        move.b  #$fb,(a2)+  * Dataadressmark
        move.w  drbyte,d0   * Anzahl der Byte/Sektor als Zähler
        move.b  #$e5,d7     * für die Datenbyte dieses Sektors
        jsr     wpuff       * $E5 als Datenbyte in Puffer schr.
        move.b  #$f7,(a2)+  * Checksum schreiben
```

```
        move.w   gap4,d0         * gap5 mal $4E als Füllbyte für Lücke4
        move.w   #$4e,d7
        jsr      wpuff           * in Puffer schreiben
        move.w   sektor,d0       * momentanen Sektor um eins erhöhen
        addq.w   #1,d0
        move.w   d0,sektor
        cmp.w    asector,d0      * mit Anzahl der Sektoren pro Track
        ble      makt1           * vergleichen, wenn größer, dann
        move.w   gap5,d0         * ist der letzte Sektor in den Puffer
        move.w   #$4e,d7         * geschrieben, und nun kann die 5.
        jsr      wpuff           * Lücke am Trackende mit $4E gefüllt
        rts                      * werden, dann zurück

**************************************************************************
*  schreibt den Byte-Wert in Register D7, D0-mal in den durch            *
*  Adressregister A2 adressierten Puffer                                 *
**************************************************************************

wpuff:   subq.w   #1,d0          * Zähler anpassen
wpuff1:  move.b   d7,(a2)+       * in Puffer schreiben
         dbra     d0,wpuff1      * D0-mal
         rts


**************************************************************************
*  übergibt die Adresse des Trackpuffer an den DMA-Controller, die      *
*  Routine muß im Supervisor-Mode aufgerufen werden.                     *
**************************************************************************

setbuf:  move.l   #formbuf,d0    * Adresse des Trackpuffers
         move.b   d0,dmalow      * Low-Byte eintragen
         lsr.l    #8,d0          * 8-Bit rechtschieben
         move.b   d0,dmamid      * und nächstes Byte eintragen
         lsr.l    #8,d0          * acht Bit rechtsschieben
         move.b   d0,dmahigh     * und Highbyte eintragen
         rts
```

```
**********************************************************************
*  formattiert einen Track, indem der Inhalt des Trackpuffers        *
*  (in formbuf) durch den write-Track Befehl des Diskcontrollers     *
*  direkt auf die Diskette geschrieben wird                          *
**********************************************************************


xfortrac: move.w  #$190,dmamode   * DMA löschen und auf schreiben
          move.w  #$90,dmamode    * umschalten
          move.w  #$190,dmamode
          move.w  #$1f,d6         * 31 ins Sektorcountregister ein-
          jsr     wrcontr         * tragen
          move.w  #$180,dmamode   * FDC-Register selektieren
          move.w  #$f8,d6         * write-Track Befehl
          jsr     wrcontr
          move.l  #$60000,d7      * Time-out Zähler
xfort1:   subq.l  #1,d7           * erniedrigen
          beq     xforterr        * wenn abgelaufen, dann Fehler
          btst    #5,mfp          * FDC schon fertig?
          bne     xfort1          * wenn nicht, weiter warten
          rts                     * sonst zurück

xforterr: move.w #-24,-(a7)       * Fehlernummer auf Stack und
        jsr     errhand           * handlen
        rts

**********************************************************************
*  ruft die Routine zum direkten Formattieren eines Tracks auf       *
**********************************************************************


xformat: movem.l a3-a6/d3-d7,-(a7)
        jsr     cursmess
        jsr     delline       * Register retten und Message
        move.l  #xffrag1,a0   * ausgeben
        jsr     printf
        jsr     leerebuf      * Tastaturbuffer leeren und
        jsr     wtast         * auf Tastendruck warten
        cmp.b   #'y',d0
        beq     xformit
```

```
          cmp.b    #'Y',d0
          bne      xformend       * weder groß noch klein 'y'

xformit:  jsr      super          * sonst Supervisor on
          st       flock          * Floppy-Interrupt sperren
          jsr      setplatz       * einmal den Track lesen, dient
          jsr      seldrive       * der Beschleunigung der Diskette
          jsr      flreset        * da sonst bei den inneren Tracks
          jsr      searcht        * die Umfangsgeschwindigkeit der
          jsr      rdstrack       * Diskette nicht ausreichend ist
          jsr      setbuf         * jetzt den Trackpuffer an DMA-Co.
          jsr      maketr         * den aktuellen Track im Puffer er-
          jsr      searcht        * zeugen, und den Track suchen
          jsr      xfortrac       * nun den Track auf Diskette schreiben
          jsr      leerebuf
          jsr      cursmess
          jsr      delline
          jsr      flreset        * Controller resetten
          jsr      user           * User-Mode wieder einschalten
          move.l   #xffrag2,a0
          jsr      printf         * Message ausgeben
          jsr      wtast          * Tastatur abfragen
          jsr      super          * Supervisor einschalten
          sf       flock          * Floppy-Interrupt freigeben
          jsr      deselect       * Laufwerk deselektieren
          jsr      user           * User-Modus einschalten

xformend: jsr      cursmess
          jsr      delline        * Message ausgeben, Register
          jsr      cursmess       * zurückholen, und
          move.l   #drfrag1,a0
          jsr      printf
          movem.l  (a)+,a3-a6/d3-d7
          rts                     * zurück


*************************************************************************
*  die folgenden Menuepunkte ermöglichen das Erhöhen und Erniedrigen  *
```

```
*  der gaps im Menue, Einzelheiten bitte beim Sektormenue nach-        *
*  schlagen.                                                           *
************************************************************************

incgaps: cmp.w   #99,d0      * maximal Anzahl der Füllbyte für
         blt     incgaps1    * alle Gaps gleich 99, diese Routine
         move.w  #0,d0       * wird von allen incgap-Menuepunkten
         bra     incgaps2    * aufgerufen, da die Begrenzungen
incgaps1: addq.w #1,d0       * gleich sind
incgaps2: rts


incgap1: move.w  gap1,d0
         jsr     incgaps
         move.w  d0,gap1
         divu    #10,d0
         add.b   #'0',d0
         move.b  d0,mgap1
         swap    d0
         add.b   #'0',d0
         move.b  d0,mgap1+1
         jsr     dispmen
         rts

incgap2: move.w  gap2,d0
         jsr     incgaps
         move.w  d0,gap2
         divu    #10,d0
         add.b   #'0',d0
         move.b  d0,mgap2
         swap    d0
         add.b   #'0',d0
         move.b  d0,mgap2+1
         jsr     dispmen
         rts

incgap3: move.w  gap3,d0
         jsr     incgaps
         move.w  d0,gap3
         divu    #10,d0
```

```
        add.b    #'0',d0
        move.b   d0,mgap3
        swap     d0
        add.b    #'0',d0
        move.b   d0,mgap3+1
        jsr      dispmen
        rts

incgap4: move.w  gap4,d0
        jsr      incgaps
        move.w   d0,gap4
        divu     #10,d0
        add.b    #'0',d0
        move.b   d0,mgap4
        swap     d0
        add.b    #'0',d0
        move.b   d0,mgap4+1
        jsr      dispmen
        rts

incgap5: move.w  #-1,-(a7)
        move.w   #11,-(a7)          * Keyboard-Shift gedrückt ?
        trap     #13
        addq.l   #4,a7
        move.w   #10,d1
        btst     #0,d0
        bne      incgap5x
        btst     #1,d0
        bne      incgap5x
        move.w   #1,d1
incgap5x: move.w  gap5,d0
        add.w    d1,d0
        cmp.w    #999,d0
        ble      incgap5a
        move.w   #0,d0
incgap5a: move.w  d0,gap5
        ext.l    d0
        divs     #100,d0
        add.b    #'0',d0
        move.b   d0,mgap5
```

```
        swap    d0
        ext.l   d0
        divs    #10,d0
        add.b   #'0',d0
        move.b  d0,mgap5+1
        swap    d0
        add.b   #'0',d0
        move.b  d0,mgap5+2
        jsr     dispmen
        rts

**********************************************************************

decgaps: cmp.w  #0,d0           * wird von allen decgap Menuepunkten
        ble     decgaps1        * aufgerufen, da max und min Anzahl
        subq.w  #1,d0           * bei allen gabs gleich ist
        bra     decgaps2
decgaps1: move.w #99,d0
decgaps2: rts

decgap1: move.w gap1,d0
        jsr     decgaps
        move.w  d0,gap1
        divu    #10,d0
        add.b   #'0',d0
        move.b  d0,mgap1
        swap    d0
        add.b   #'0',d0
        move.b  d0,mgap1+1
        jsr     dispmen
        rts

decgap2: move.w gap2,d0
        jsr     decgaps
        move.w  d0,gap2
        divu    #10,d0
        add.b   #'0',d0
        move.b  d0,mgap2
        swap    d0
        add.b   #'0',d0
```

```
        move.b   d0,mgap2+1
        jsr      dispmen
        rts

decgap3: move.w  gap3,d0
        jsr      decgaps
        move.w   d0,gap3
        divu     #10,d0
        add.b    #'0',d0
        move.b   d0,mgap3
        swap     d0
        add.b    #'0',d0
        move.b   d0,mgap3+1
        jsr      dispmen
        rts

decgap4: move.w  gap4,d0
        jsr      decgaps
        move.w   d0,gap4
        divu     #10,d0
        add.b    #'0',d0
        move.b   d0,mgap4
        swap     d0
        add.b    #'0',d0
        move.b   d0,mgap4+1
        jsr      dispmen
        rts

decgap5: move.w  #-1,-(a7)
        move.w   #11,-(a7)
        trap     #13
        addq.l   #4,a7
        move.w   #10,d1
        btst     #0,d0
        bne      decgap5x
        btst     #1,d0                * rechte Shifttaste
        bne      decgap5x
        move.w   #1,d1
```

```
decgap5x: move.w   gap5,d0
        sub.w     d1,d0
        bpl       decgap5a
        move.w    #999,d0

decgap5a: move.w   d0,gap5
        ext.l     d0
        divs      #100,d0
        add.b     #'0',d0
        move.b    d0,mgap5
        swap      d0
        ext.l     d0
        divs      #10,d0
        add.b     #'0',d0
        move.b    d0,mgap5+1
        swap      d0
        add.b     #'0',d0
        move.b    d0,mgap5+2
        jsr       dispmen
        rts


*************************************************************************
*  verändert die Anzahl der Byte pro Sektor, diese wird in drbyte      *
*  gespeichert und beeinflußt auch die Anzahl der angezeigten und      *
*  geschriebenen Byte im Sektormenue, natürlich nur wenn das Format    *
*  Modul eingebaut ist.                                                *
*************************************************************************

incbyte: move.w    drbyte,d0        * mögliche Anzahl der Byte/Sektor
        cmp.w     #128,d0          * ist 128, 256, 512 oder 1024 Byte
        beq       incby1
        cmp.w     #256,d0
        beq       incby2
        cmp.w     #512,d0
        beq       incby3
        move.w    #128,d0
        move.b    #'0',mdrisekt     * Auch in den Menuetext eintragen
        move.b    #'1',mdrisekt+1
        move.b    #'2',mdrisekt+2
```

```
         move.b   #'8',mdrisekt+3
         bra      incbywei

incby1:  move.w   #256,d0
         move.b   #'0',mdrisekt
         move.b   #'2',mdrisekt+1
         move.b   #'5',mdrisekt+2
         move.b   #'6',mdrisekt+3
         bra      incbywei

incby2:  move.w   #512,d0
         move.b   #'0',mdrisekt
         move.b   #'5',mdrisekt+1
         move.b   #'1',mdrisekt+2
         move.b   #'2',mdrisekt+3

         bra      incbywei
incby3:  move.w   #1024,d0
         move.b   #'1',mdrisekt
         move.b   #'0',mdrisekt+1
         move.b   #'2',mdrisekt+2
         move.b   #'4',mdrisekt+3

incbywei: move.w  d0,drbyte
         jsr      dispmen
         rts


*************************************************************************
*  erniedrigt die Anzahl der Byte/Sektor, läßt ebenso wie incbyte nur *
*  die vier möglichen Werte des FDC (128, 256, 512, 1024) zu          *
*************************************************************************

decbyte: move.w   drbyte,d0
         cmp.w    #128,d0
         beq      decby1
         cmp.w    #256,d0
         beq      decby2
         cmp.w    #512,d0
         beq      decby3
```

```
        move.w   #512,d0
        move.b   #'0',mdrisekt
        move.b   #'5',mdrisekt+1
        move.b   #'1',mdrisekt+2
        move.b   #'2',mdrisekt+3
        bra      decbywei

decby1: move.w   #1024,d0
        move.b   #'1',mdrisekt
        move.b   #'0',mdrisekt+1
        move.b   #'2',mdrisekt+2
        move.b   #'4',mdrisekt+3
        bra      decbywei

decby2: move.w   #128,d0
        move.b   #'0',mdrisekt
        move.b   #'1',mdrisekt+1
        move.b   #'2',mdrisekt+2
        move.b   #'8',mdrisekt+3
        bra      decbywei

decby3: move.w   #256,d0
        move.b   #'0',mdrisekt
        move.b   #'2',mdrisekt+1
        move.b   #'5',mdrisekt+2
        move.b   #'6',mdrisekt+3

decbywei: move.w d0,drbyte
        jsr      dispmen
        rts
```

*Die Bedienung des Diskeditors*

Die Steuerung des Diskeditors erfolgt fast ausschließlich mit den Cursortasten; Cursor links und rechts wählen die verschiedenen Menüpunkte an, Cursor hoch und runter selektieren diesen Menüpunkt oder verändern den variablen Teil eines Menüpunktes (drive, side, track ect.). Bei manchen Menüpunkten führt die

Auswahl in ein neues Menü, aus dem dann wieder mit Cursor links und rechts ein neuer Menüpunkt ausgesucht werden kann.

Und hier nun die Erklärung der einzelnen Menüpunkte:

### 7.2.1 Das Hauptmenü

Alle Punkte des Hauptmenüs, mit der Ausnahme des 'Ende'-Punktes, führen in ein neues Menü:

TRACK: wählt das Track-Menü aus, in dem die Behandlung ganzer Tracks möglich ist.

TRACK/SYNC: Auswahl des Track-with-Syncbytes-Menüs; in diesem Modus kann man auf alle Informationen der Diskette zugreifen, z.B. auf die Gap- und Synchronationsbytes.

SEKTOR: Auswahl des Sektor-Mode, der das Lesen, Editieren und Schreiben von Sektoren ermöglicht.

CLUSTER: Auswahl des Cluster-Modus, in dem über Cluster auf die Diskette zugegriffen werden kann.

FORMAT: Auswahl des Format-Modus, der das Formatieren einzelner Tracks mit verschiedenen Formaten, auch Nicht-ATARI-Formaten, erlaubt.

OPTIONS: Auswahl des Options-Menüs, in dem das aktuelle Laufwerk neu bestimmt werden kann. Außerdem kann man die maximale Anzahl der Tracks und Sektoren festlegen.

ENDE: Beendet das Programm und kehrt zum Desktop zurück

### 7.2.2 Das Track-Menü

Das Track-Menü enthält selbst wieder eine Anzahl von Menüpunkten:

drive: 0 wurde dieser Punkt ausgewählt, kann man mit Cursor-up und down das aktuelle Drive neu anwählen. Alle Menüpunkte, denen ein Doppelpunkt und eine Zahl folgt, bieten diese Möglichkeit des Veränderns durch Betätigung der Cursor-Tasten.

side 0: Auswahl der Laufwerksseite, 0 oder 1.

track 00: Auswahl des Tracks auf den dann zugegriffen wird. Die maximale auswählbare Nummer wird im Option-Menü des Hauptmenüs bestimmt.

sect/trac 00: bestimmt die Anzahl der zu lesenden und schreibenden Sektoren pro Track.

READ: durch Anwahl dieses Punktes wird der augenblicklich angezeigte Track des aktuellen Drives gelesen und anschließend angezeigt. Mit Cursor-up und down kann man durch die einzelnen Sektoren dieses Tracks scrollen. Es ist auch möglich, während des Scrollens durch den Text durch Betätigen von Cursor-rechts und links die Edit-Funktion aufzurufen und so im Track etwas zu ändern.

WRITE: nun wird der gesamte Track nach einer Sicherheitsabfrage wieder auf die Diskette zurückgeschrieben.

EDIT: bietet die Möglichkeit des Editierens eines Sektors des Tracks. Es kann aus der Read-Funktion aufgerufen werden, man kann jeweils nur einen Sektor des Tracks editieren.

BACK: kehrt zurück zum Hauptmenü

### 7.2.3 Das Track with Syncs-Menü

Hier wieder die einzelnen Unterpunkte:

drive 0: Auswahl des aktuellen Laufwerks

side 0: Auswahl der aktuellen Seite

track 00: Auswahl des aktuellen Tracks

readwithsync: liest den gesamten Track mit allen Gaps und ermöglicht das Scrollen durch diesen Track.

Addrfield: zeigt die Adreßfelder des gesamten Tracks mit Bytegröße und Checksumme an. Die Ausgabe erfolgt teilweise doppelt, da immer 16 Adreßfelder angezeigt werden.

BACK: Zurück zum Hauptmenü

### 7.2.4 Das Sektor-Menü

Die einzelnen Menüpunkte des Sektor-Menüs:

drive 0: Auswahl des aktuellen Laufwerks

side 0: Auswahl der aktuellen Seite

track 00: Auswahl des aktuellen Tracks

sektor 00: Auswahl des aktuellen Sektors

READ: Lesen des aktuellen Sektors und Anzeigen desselben

WRITE: Schreiben des im Speicher befindlichen Sektors in den momentan angezeigten Sektor nach Sicherheitsabfrage.

EDIT: verzweigt in den Edit-Modus, der die Eingabe von Sedezimal-Zahlen erlaubt und bei dem man mit den Cursor-Tasten alle Bytes des Sektors ansteuern kann. Der Edit-Modus wird durch Betätigung der Return-Taste wieder verlassen.

BACK: Zurück zum Hauptmenü

### 7.2.5 Das Cluster-Menü

Die einzelnen Menüpunkte des Cluster-Menüs:

drive 0: Auswahl des aktuellen Laufwerks

clust 0000: Auswahl des aktuellen Clusters, geschieht wiederum durch Cursor Up und Down, gleichzeitiges Betätigen der SHIFT-Taste und Cursor UP und Down, erhöht bzw. ernidrigt die Clusternummer um 10.

READ: liest den aktuellen Cluster in den Speicher ein und berechnet den physikalischen Sektor nebst Track und Seite, an dem dieser Cluster beginnt. Die Informationen über den physikalischen Sektor werden ins Sektor-Menü eingetragen, d.h. wenn man nach dem Lesen eines Clusters ins Sektor-Menü überwechselt, kann man den Startsektor dieses Cluster durch READ sofort lesen.

NEXT: berechnet mit Hilfe des File-Allocation-Table den Nachfolgecluster innerhalb des Files und liest ihn in den Speicher ein.

WRITE: schreibt den im Speicher befindlichen Cluster nach einer Sicherheitsabfrage auf den aktuellen, im Menü angezeigten, Cluster der Diskette.

EDIT: ermöglicht das Editieren eines Clusters, 'Return' beendet den Edit-Modus.

Startoffile: Zeigt alle Files des aktuellen Laufwerks mit zugehörigen File-Attributen. Das Scrollen durch die einzelnen Files ermöglichen wiederum die Cursor-up und down Tasten, wobei ein Druck auf die Return-Taste den Startcluster des momentan angewählten Files in das Cluster-Menü überträgt. Sollte das angewählte File ein Subdirectory repräsentieren, wird in dieses verzweigt und man kann nun ein File des Subdirectories anwählen. Zurück ins Hauptdirectory (Rootdirectory) gelangt man durch Anwahl des einzelnen Punktes.

BACK: zurück zum Hauptmenü.

### 7.2.6 Das Format-Menü

Die Punkte des Format-Menüs:

drive 0: Auswahl des aktuellen Laufwerks

side 0: Auswahl der aktuellen Seite

track 00: Auswahl des aktuellen Tracks

sec/tra.00: Auswahl der Sektoren pro Track. Die maximal mögliche Anzahl hängt von dem Menüpunkt MAXSEKT im Options-Menü ab, d.h. steht bei MAXSEKT eine 10, kann man auch bei sec/track die 10 anwählen.

FORMAT: Formatiert den aktuellen Track mit sec/track Sektoren nach vorheriger Sicherheitsabfrage. Die Formatierung erfolgt durch die XBIOS-Funktion und formatiert grundsätzlich Sektoren mit 512-Bytes (ATARI-Format).

XFORMAT: Formatiert den aktuellen Track mit den im GAP-Menü veränderbaren Parametern, so kann man z.B. die Anzahl der Bytes/Sektor, aber auch die Anzahl der Synchronisationsbytes selbst bestimmen.

GAPS: Verzweigt in ein eigenes Menü, in dem dann die Parameter für XFORMAT eingestellt werden können.

BACK: Zurück zum Hauptmenü

### 7.2.7 Das GAP-Menü

Die Punkte des Gap-Menüs:

GAP1 00: bestimmt die Anzahl der Füllbytes am Trackanfang, maximal möglich sind 99.

GAP2 00: bestimmt die Anzahl der Null-bytes

GAP3 00: bestimmt die Anzahl der

GAP4 00: bestimmt die Anzahl der

GAP5 00: bestimmt die Anzahl der Füllbyte, die am Track-Ende eingefügt werden.

BYT/SEK: hier können die vier vom Diskcontroller unterstützten Byte pro Sektor Formate (128, 256, 512, 1024) eingestellt werden. Die Auswahl beeinflußt auch das Lesen eines Sektors im Sektor-Menü, da bei 1024-Byte Sektoren auch die Möglichkeit bestehen muß, diese zu Editieren ect.

BACK: zurück zum Hauptmenü

### 7.2.8 Das Options-Menü

Die Punkte des Options-Menüs:

drive 0: Auswahl des aktuellen Laufwerks

MAXTRACK 00: Auswahl des maximalen anwählbaren Tracks für die Track-Menüpunkte.

MAXSEKT 00: Auswahl des maximalen anwählbaren Sektors für die Sektor-Menüpunkte, bestimmt außerdem auch die maximal anwählbaren Sektoren pro Track.

INIT DRIVE: Aufruf der BIOS-Funktion, Anzeige der Disk-Parameter

SHOW BPB: Zeigt die Bios-Parameterblock des aktuellen Laufwerks.

BACK: Zurück zum Hauptmenü

## 7.3 Beispiele zur Benutzung des Disk-Editors

Als erstes wollen wir den Diskeditor einmal am Directory und an der File Allocation Table (FAT) praktisch erproben. Zu diesem Zweck formatieren Sie bitte eine einseitige Diskette mit dem Namen "WORK.TST" und kopieren auf diese das "DESK2.ACC" Accessory von der Systemdisk, das eine Länge von 6258 Bytes hat (neuestes TOS).

Wie schon vorher erwähnt, teilt das GEMDOS die Disketten in Blöcke (Cluster) zu zwei Sektoren mit jeweils 512 Bytes, also insgesamt 1024 Bytes, ein. Sechs mal 1024 ergibt 6144, das "DESK2.ACC" paßt folglich nicht in 6 Cluster, sondern belegt auch noch einige Byte des 7. Diskettenclusters. Nun laden Sie bitte den Diskeditor, wählen das Sektor-Menü und stellen Drivenummer, Track 1, Seite 0 und Sektor 3 ein. Auf diesem Sektor steht auf einseitig formatierten Disketten der erste

und betrachten das angezeigte Ergebnis. Mit der Taste 'p' können Sie sich den Sektor auch ausdrucken lassen.

Sehen wir uns nun das Ergebnis Ihrer Aktionen an. Sie finden als erstes den bei der Formatierung gewählten Namen wieder. Jeder Directory-Eintrag, der Diskettenname zählt auch als solcher, belegt genau 32 Bytes. Der Diskname benötigt die Bytes 0 bis 31 des Sektors und der erste Directory-Eintrag beginnt bei Byte 32 (Sedezimal $20). Die ersten elf Bytes (0 bis 10) jedes Directory-Eintrags sind für den Filenamen reserviert, und zwar wird nicht benötigter Platz der acht möglichen Buchstaben des Filenamens durch Spaces (Sedezimal $20) belegt. Die nächsten drei Bytes (8-10) belegt die File-Extension.

Alle nun folgenden Daten stehen im Intel-Format auf der Diskette, d.h. belegen die Daten mehr als ein Byte, steht stets das Low-Byte vor dem High-Byte. Das Datenwort $1234 wäre somit als $34 $12 gespeichert.

Das zwölfte Byte (Nummer 11) jedes Eintrags fungiert als Dateiattribut und kennzeichnet die verschiedenen Zugriffsmöglichkeiten auf das File. Die Zahl $08 im Dateiattributfeld des Diskettennamens deklariert diesen Eintrag als Diskettennamen.

Nun folgen 11 Bytes (Nummer 12 bis 21, $0C-$15) die ohne Bedeutung sind, doch in Byte 22 und 23 ($16 $17), relativ zum Eintragsanfang, steht die Uhrzeit des letzten Schreib-Zugriffs auf das File. Die Uhrzeit ist in den Bits dieser beiden Bytes Kodiert, wie Sie ja bereits im Kapitel 3.3 erfahren haben. Als Besonderheit ist noch anzumerken, daß die Sekunden nur im 2-Sekundentakt gezählt werden, so daß die Sekunden noch mit zwei multipliziert werden müssen.

Im Anschluß an die Uhrzeit-Bytes folgen im Directory-Eintrag die Datum-Bytes (24 - 25, $18 $19,) die in ähnlicher Weise codiert sind.

Womit wir nun schon zu den beiden wichtigsten Bytes eines jeden Directory-Eintrages kommen, zu den Bytes 26 und 27 ($1A, $1b) relativ zum Eintragsanfang, die den Startcluster des

Files angeben. Wie Sie selbst sehen, steht bei "unserem" File $02 $00, was umgeformt auch schon den Startcluster $0002 ergibt. Das File "DESK2.ACC" beginnt also bei Cluster Nummer 2, welcher gleichzeitig für das Betriebssystem den ersten freien Cluster einer Diskette darstellt. Zur Umrechnung der Clusternummer in die logische und physikalische Sektornummer müssen einige Parameter des BIOS-Parameterblockes und des Bootsektors zu Rate gezogen werden (clsize,datrec,spt,nside). Die Umrechnung geschieht folgendermaßen:

1. Subtraktion von 2 von der Clusternummer, da Cluster Nummer 2 laut Zählweise des Betriebsystems der erste freie Cluster ist.

2. Multiplikation des obigen Resultates mit der Anzahl der Sektoren pro Cluster (clsize = 2 bei ATARI ST).

3. Addition der Nummer des ersten logischen Datensektors, (datrec = 18 bei ATARI ST)

Die erhaltene Zahl repräsentiert die logische Sektornummer des ersten Sektors dieses Clusters. Der zweite Sektor des Clusters ist der direkt folgende (bei einseitigen Disketten). Bei zweiseitig formatierten Disketten befindet sich der erste Sektor des ersten Datenclusters auf Seite 0, Track 1, Sektor 1, der zweite Sektor dieses Clusters ist der Sektor 2 auf Track 1 Seite 0. Die Cluster werden also auch in diesem Fall durch hintereinanderliegende Sektoren auf einem Track gebildet. Wird allerdings der letzte Sektor eines Tracks auf Seite 0 erreicht, schreibt das Betriebssystem den Folgesektor auf den gleichen Track, Sektor 1 der Seite 1. Wurden also wie im ATARI-Format Tracks mit 9 Sektoren formatiert, so liegt der erste physikalische Sektor des fünften Datenclusters auf Seite 0, Track 1, Sektor 9 und der zweite Sektor dieses fünften Datenclusters wird auf Seite 1, Track 1, Sektor 1.

Doch nun zurück zum logischen Sektor des Startclusters. Es fehlt nämlich noch die Berechnung von physikalischem Track und Sektor, auf dem sich dieser logische Sektor, dessen Zählweise ja bei null beginnt, befindet. Wir nehmen wieder eine einseitige

Diskette als Grundlage und teilen nun die logische Sektornummer durch die Anzahl der Sektoren pro Track (spt = 9, ATARI-Format): das Ergebnis der Division ist der Track, der Rest der Division der Offset zum Sektor 1 dieses Tracks.

Wieder auf das erste File angewandt: Cluster Nummer 2 minus 2 ergibt null, mal 2 Sektoren pro Cluster bleibt null, plus 18 gleich 18, geteilt durch 9 Sektoren pro Track gleich 2 Rest Null: Der erste Sektor des ersten Clusters vom File "DESK2.ACC" ist somit auf Track 2 Sektor 1 lokalisiert.

Für eine zweiseitig formatierte Diskette läuft die Reohnung bis zum logischen Sektor gleich, nur die Umrechnung auf physikalischen Track und Sektor muß nun die Seite mitberücksichtigen: die logische Sektornummer wird wieder durch die Anzahl der Sektoren pro Track (spc) dividiert, und der Rest dieser Division ergibt auch den Offset zum Sektor 1, nur die Berechnung des Tracks differiert ein wenig. Das Ergebnis der Division durch die Anzahl der Sektoren pro Track wird durch die Anzahl der Seiten, nämlich 2 bei doppelseitig formatierten Disketten, geteilt; ist die Division ohne Rest möglich, so repräsentiert das Ergebnis den Track für diesen Sektor auf Seite null der Diskette. Entsteht bei der Division ein Rest, so befindet sich der berechnete Track auf Seite 1 der Diskette.

Nun zur Berechnung des physikalischen Sektors für den Cluster 2 bei doppelseitig formatierten Disketten. An der Berechnung des logischen Sektors ändert sich nichts: hier ergibt sich auch 18, nun dividiert man durch die Anzahl der Sektoren pro Track (9) und erhält 2 Rest 0, die Sektornummer ist also 1 (wegen Nullrest). Zur Ermittelung von Track und Seite dividiert man nun die eben gewonnene zwei durch die Anzahl der Seiten (2) der Diskette, dies ergibt 1 Rest 0. Somit befindet sich der Sektor, nach oben genannter Regel, auf Track 1 Seite 0 der Diskette.

Zur weiteren Übung wollen wir noch die Startsektor des Directories für ein- und zweiseitig formatierte Disketten ermitteln. Der logische Startsektor ergibt sich wieder aus BPB und Bootsektor, indem man die Anzahl der reservierten Sektoren (res = 1 bei ATARI) zu dem Produkt aus Anzahl der File-Allocation-

Tables (fat = 2) und Anzahl der Sektoren pro File-Allocation-Table (spf = 5) addiert. Also logischer Sektor = 1 + 2 * 5 = 11. Der Directory-Anfang befindet sich auf allen ATARI-Disketten auf dem logischen Sektor 11. Für eine einseitig formatierte Diskette folgt für den physikalischen Sektor:

11 geteilt durch 9 (Sektoren/Track) gleich 1 Rest 2. Sektor gleich 1 plus Rest = 3, der logische Sektor 11 befindet sich also auf einseitigen Disketten auf Track 1. Sektor 3. Für eine doppelseitige Diskette:

11 geteilt durch 9 gleich 1 Rest 2, Sektor gleich 1 plus Rest = 3, 1 dividiert durch 2 ergibt 0 Rest 1, es existiert ein Rest => Seite 1, Track gleich Ergebnis der ersten Division (1). Der logische Sektor 11 befindet sich auf doppelseitigen Disketten auf Seite 1, Track 0, Sektor 3, wie Sie selbst feststellen können.

Bei der Clusterberechnung haben wir den Directory-Eintrag für das File "DESK2.ACC" vollkommen aus den Augen verloren, es fehlen nämlich zu den 32 Bytes pro Directory-Eintrag nur noch die letzten vier Bytes (28-31, $1C-$1F), welche die Dateigröße in Byte angeben. Für $72 $18 $00 $00 ergibt sich $00001872 (Intel läßt grüßen), und diese Sedezimal-Zahl stimmt nach der Umwandlung ins Dezimalsystem (6258) auffallend mit der vom Desktop angezeigten Größe überein. Bevor Sie sich nun das eben gesagte noch einmal in Ruhe in einer Tabelle anschauen können, müssen noch zwei Bytes mit Sonderfunktionen erläutert werden: es sind dies das erste Byte des Directory-Eintrages (Byte 0) sowie das zwölfte Byte (Byte 11), das Attribut-Byte.

Das erste Byte des Namens, im Beispiel $44, bedeutet bei Bytes ungleich $E5, $00 und $2E den ASCII-Code des ersten Buchstabens ($44 = 'D'). Die anderen Einträge haben folgende Bedeutung:

*1. Byte des Namens: Bedeutung:*

$00: iese Datei wurde noch nicht benutzt (trivial $E5 : Diese Datei wurde zwar schon benutzt, ist aber gelöscht worden.

$2E: Dieses Byte weist auf den Pfad vom Subdirectory zum Rootdirectory. Ist das nächste Byte ebenfalls $2E, so steht im Feld der Clusternummer die Clusternummer des nächsten übergeordneten Directories, ist das zweite Byte $00, so ist das übergeordnete Directory das Rootdirectory. Dieser Zusammenhang wird weiter unten noch genauer erklärt.

Das Attribut-Byte (Byte 11) kann folgende Werte annehmen:

$00: Diese Datei (File) kann sowohl gelesen als auch beschrieben werden.
$01: Diese Datei kann nur gelesen werden (read only).
$02: Diese Datei wird im Directory nicht angezeigt (hidden).
$08: Dies kennzeichnet den Diskettennamen, alle Bytes nach dem 10 haben keine weitere Bedeutung.
$10: Bei dem Dateinamen handelt es sich um ein Subdirectory (Ordner)

Die Bedeutung der 32 Bytes jedes Directory-Eintrages ist:

| Byte: | Bedeutung |
|---|---|
| 0-10 | Dateinammen mit Extension, erstes Byte ist evtl. der Status ($E5,$2E) |
| 11 | Dateiattribut, (read/write, readonly, subdirectory) |
| 12-21 | unbenutzt |
| 22-23 | Uhrzeit |
| 24-25 | Datum |
| 26-27 | Startcluster des Files |
| 28-31 | Filegröße in Bytes |

### 7.3.1 File-Allocation Table

Sie wissen nun, wie sie den Anfang eines Files auf der Diskette finden können: einfach die Bytes Nummer 26 und 27 des Directory-Eintrages für das File ins Dezimalsystem umwandeln, die so erhaltene Clusternummer nach oben erläuterten Regeln in logische Sektornummer ect. umwandeln. Doch wo befindet sich der zweite Cluster eines Files, wo der letzte?

Alle diese Fragen klärt die File-Allocation-Table, deren Anfang immer auf Seite 0, Track 0, Sektor 2 zu finden ist (single und double sided). Zum besseren Verständnis lesen Sie doch einfach den Sektor 2 auf Track 0, Seite 0 ihrer einseitig formatierten Diskette mit dem File (Sie können den Filenamen bestimmt schon auswendig) mittels Diskeditor und 'read' in den Speicher. Von den ersten drei Bytes einmal abgesehen ($F7 $FF $FF) ist doch schon eine Struktur zu erkennen.

In dieser FAT befinden sich Informtionen über jeden Cluster der Diskette, und zwar, ob er belegt ist, und wenn ja, welches der nächste Cluster des Files ist. Lassen wir erst einmal die seltsame Struktur (03 40 00 05) außer acht und nehmen als Ziffernfolge 3, 4, 5 an. Nehmen wir weiter an, die Tabelle sehe folgendermaßen aus:

START: 1, 2, 3, 4, 5

Start soll nur die Adresse symbolisieren, an der sich die Zahl Eins befindet. Das ganze ist eine lineare Liste, das nullte Listenelement (relativ zu Start) hat den Wert 1, das erste den Wert 2 ect. Lesen wir nun die Adresse Start, so finden wir den Wert Eins. Diese Zahl Eins soll die Information des nächsten zu lesenden Listenelementes darstellen. Zur Adreßermittlung addieren wir die Zahl Eins zur Adresse von Start und lesen die nun gefundene Adresse (START+1) aus, was als Ergebnis die Zahl zwei liefert.

Diese zwei sagt uns also: die Nummer des nächsten zu lesenden Listenelementes befindet sich an der Adresse START+2, hier steht die Zahl 3. So wird es möglich sich von Listenelement zu

Listenelement zu hangeln, indem man einfach die Elemente liest und das Ergebnis als Offset relativ zum Start der Liste ansieht. Dies klingt auf den ersten Blick ein wenig unverständlich, ist aber genau die Methode, mit der das Betriebssystem die Cluster der Diskette verwaltet.

Sehen wir uns also noch einmal die vereinfachte Eintragung in unserer FAT (3,4,5,) an und bauen vor diese 3 Elemente 2 dummy-Eintragungen (x,x,3,4,5). Unsere Clusternummer für das File "DESK2.ACC" ist 2, das Lesen der Adresse START+0 ergibt x, ebenso das Lesen der Adresse START+1. Lesen wir aber die Adresse START+2, so erhalten wir den Wert 3, Lesen von START+3 ergibt als Resultat 4. Für DESK.ACC würde das bedeuten, der nächste auf Cluster 2 folgende Cluster ist die Nummer 3, danach folgt Nummer 4 ect.

Das Umrechnen von Clusternummer in Setornummer ist ja nun bekannt, wir benötigen nur noch eine Methode, das Ende eines Files zu kennzeichnen. Die Lösung des Betriebssystems mit Hilfe der FAT: Ergibt sich beim Lesen einer soeben gewonnenen Adresse ein bestimmter Wert, so ist der soeben gelesene Cluster der letzte des Files.

Damit das ganze noch ein wenig komplexer wird, kommt jetzt das Intel-Format in Verbindung mit einer 12-Bit Darstellung ins Spiel. Ein FAT-Eintrag belegt nämlich genau 12-Bits, das sind 3 Nibbel a 4 Bits. Die 12 Bit reichen vollkommen aus, da nicht mehr als 2^12 = 4096 Cluster auf der Diskette vorhanden sind.

Zur Erklärung des seltsamen Formates und zum Erkennen der Clusternummer werfen Sie doch bitte einen Blick auf die FAT. Denken wir uns erst einmal 16 Bits pro FAT-Eintrag, die im Intel-Format hintereinander stehen. Bei diesen 16 Bits ist das most significant Nibble (die ersten vier Bit) des High Byte überflüssig und durch ein X gekennzeichnet. Dieses freie Nibble wird nun durch das least significant Nibble (die letzten vier Bit) des Low-Byte des nächsten Eintrages belegt. Das most significant Nibble des Low-Byte und das least significant nibbel des High-Byte nehmen nach einem Überkreuztausch zusammen den Platz des Low-Bytes ein. Aus zwei 16-Bit-Einträgen wurden so

zwei 12-Bit-Einträge, wodurch ein Nibble einspart wird (das High-Byte des zweiten Eintrages). Da pro zwei Eintragungen ein Byte eingespart wird, ist das ganze System periodisch symmetrisch, d.h. nach zwei 12-Bit-Einträgen hat sich die Nibble-Schieberei ausgeglichen und das Spiel beginnt von neuem.

Zur Umrechnung von 12-Bit-Einträgen im Intel-Format in lesbare 12-Bit-Einträge im Motorola-Format ist daher die Nummer des Eintrages von entscheidender Bedeutung. Man beginnt mit dem Zählen bei Null und zählt jeweils drei Nibble als einen Eintrag. Wie schon am einfachen Beispiel erläutert (x, x), ist den ersten beiden Einträgen (Nummer null und eins) in der realen FAT (F7 FF FF) keine Bedeutung zugeordnet und der erste gültige Eintrag ist der folgende, die Nummer 2. Ist die Zugriffsnummer, wie in diesem Fall (2), gerade, so findet man die beiden niedrigwertigsten Nibbles (Motorola-Format) im ersten Byte dieses Eintrages (03) und das höchstwertigste Nibble (Motorola-Format) des Folgeclusters im zweiten Nibble des Folgebytes (0), so daß als Folgecluster der Cluster Nummer 003 (Motorola-Format) ermittelt ist.

Sucht man nun den Folgecluster dieses dritten Datenclusters, so zählt man wiederum in 3er Gruppen von Anfang an und findet als vierten Eintrag (Nummer 3) die beiden Byte 40 00, wobei der Anfang der Dreiergruppe eigentlich die Null im ersten Byte ist. Bei einer ungeraden Zugriffsnummer, in diesem Fall die 3, repräsentieren die letzten beiden Nibble des zweiten Byte (00) die High-Nibble und das erste Nibble des ersten Byte das least-significant Nibble der Clusternummer: im konkreten Beispiel wird so als Folgecluster die Nummer 004 ermittelt.

### 7.3.2 Subdirectories und Ordner auf Diskette

Das Filesystem des ATARI TOS ist hierarchisch und rekursiv, d.h. man kommt von jedem beliebigen Subdirectory (Ordner) wieder zum Hauptdirectory (Wurzel- oder Root-Directory). Zum besseren Verständis legen Sie doch bitte mit dem Desktop-Menüpunkt 'Ordner anlegen' zwei Ordner mit den Namen "ORDNER1.SUB" und "ORDNER2.SUB" an, starten den Diske-

ditor und lesen aus dem Sektor-Menü heraus den Directory-Sektor (Seite 0, Track 1, Sektor 3).

An dem schon erwähnten Byte Nummer 11 des Eintrages für "ORDNER1.SUB" ($10) erkennen Sie, daß dieser Eintrag ein Subdirectory darstellt. Der Startcluster dieses Subdirectories ist der Cluster Nummer 9 (Bytes 26 und 27), der auf Track 3 Sektor 6 der einseitigen Diskette beginnt. Lesen Sie doch bitte diesen Sektor aus dem Sektor-Menü heraus in den Speicher.

Jedes Subdirectory bekommt vom Betriebssystem eigene Directory-Sektoren "spendiert". In diesen Subdirectory-Sektoren sind die beiden ersten Einträge immer belegt, auch wenn noch keine Datei ins Subdirectory eingetragen ist. Der erste Eintrag beginnt mit einem Punkt ($2E) gefolgt von Spaces, im Attribut-Byte steht als Kennzeichen des Subdirectories eine $10 und als Startcluster ist der eigene Anfang eingetragen, in diesem Fall die 9. Im zweiten Directory-Eintrag eines Subdirectories, der von zwei Punkten ($2E) eingeleitet wird, steht im Startclustereintrag (Byte 26, 27) der Startcluster des nächsttieferen Subdirectory. In unserem Fall finden Sie zwei Null-Bytes (00 00), was bedeutet: das nächsttiefere Subdirectory ist das Hauptdirectory (Rootdirectory), da ja das Subdirectory "ORDNER1.SUB" vom Hauptdirectory aus angelegt wurde.

Verlassen Sie nun bitte den Diskeditor und lassen Sie sich vom Desktop ein Inhaltsverzeichnis des Subdirectories "ORDNER1.SUB" geben (Doppelklick auf Namen). Während also nun das leere Subdirectory "ORDNER1.SUB" angezeigt wird, wählen Sie bitte mit Hilfe des Menüpunktes 'Ordner anlegen' einen neuen Ordner mit Namen "INORD1.SUB" an und starten anschließend wieder den Diskeditor. Im Hauptdirectory (Track 1, Sektor 3) hat sich nichts geändert, so daß Sie sofort den Startsektor des Subdirectories "ORDNER1.SUB" auf Track 3, Sektor 6 mit Hilfe des Sektor-Menüs anwählen können.

Der soeben innerhalb dieses Subdirectories angelegte Ordner ist, wie Sie sehen, im Directory-Sektor des Subdirectories eingetragen, und zwar findet sich als Startcluster die Nummer 11. Clus-

ter Nummer 11 beginnt auf einseitigen Disketten auf Track 4, Sektor 1, den Sie bitte wiederum einlesen.

Im ersten Eintrag des Subdirectories "inord1.sub" findet sich der rekursive Verweis auf sich selbst und im zweiten Eintrag der Verweis auf das nächsttiefere Subdirectory, das in diesem Fall dem Subdirectory "ORDNER1.SUB" entspricht, welches auf Startcluster 9 beginnt.

### 7.3.3 Formatieren im Nicht-ATARI-Format

Es gibt zwei Arten eines Nicht-ATARI-Formates, erstens die Verwendung von mehr Tracks und Sektoren als bei der Formatierung vom Desktop (80 Tracks 0-79, und 9 Sektoren pro Track 1-9) und zweitens verschiedene Anzahl von Byte/Sektor und mehr oder weniger Synchronisationsbytes zwischen den Adreß- und Daten-Feldern. Möchten Sie z.B. den Track Nummer 81 mit 10 Sektoren formatieren, wählen Sie bitte das Options-Menü an und erhöhen Sie die Variable MAXTRACK durch Anwahl mit den Cursor-Tasten auf 81. Mit dem gleichen Verfahren erhöhen Sie bitte auch die Variable MAXSECTOR auf 10. Nun verlassen Sie das Options-Menü mittels BACK und wählen das Format-Menü an. Hierin erhöhen Sie bitte die Tracknummer auf 81 und die Variable SEC/TRAC auf 10. Anschließend brauchen Sie nur noch FORMAT anzuwählen und die nun folgende Sicherheitsabfrage durch die Taste 'y' zu bestätigen: schon wird der Track 81 mit 10 Sektoren pro Track formatiert.

Möchten Sie das ATARI-Format ganz verlassen und auch die Zwischenräume zwischen den Sektoren und die Sektorgröße nach eigenen Wünschen gestalten, müssen wir die TOS-Programmierung hinter uns lassen und direkt auf den Diskcontroller zugreifen. Natürlich müssen wir uns bei der Auswahl auf die Grenzen des Diskcontrollers beschränken, so kann man z.B. nicht Sektoren mit 630 Bytes/Sektor formatieren, da der Diskcontroller nur vier verschiedene Sektorgrößen handhaben kann (128, 256, 512, 1024). Zum Formatieren des Tracks Nummer 79 mit 4 Sektoren mit jeweils 1024 Byte pro Sektor und einem Trackvorspann von 32 mal $4E anstelle von 60 mal $4E wählen

Sie bitte aus dem Menüpunkt FORMAT den Punkt GAPS aus. Sie erhalten dann ein neues Menü, wo Sie bitte den Wert für GAP1 auf 32 erniedrigen und dann BYT/SEC auf 1024 erhöhen. Nachdem Sie mit BACK wieder im Format-Menü gelandet sind, erniedrigen Sie hier bitte den Menüpunkt SEC/TRAC auf 4 und wählen letztlich den Menüpunkt XFORMAT.

Die Sicherheitsabfrage muß wieder mit 'y' beantwortet werde und schon ist der Track 79 neu formatiert. Sie sollten übrigens wirklich die geforderte Sekunde oder ein bißchen mehr warten, da sonst der Laufwerksmotor weiterläuft. Ich hätte natürlich auch eine Zwangspause einbauen können, aber in der Testphase war die Option eines laufenden Motors sehr nützlich, da die verschiedenen Laufwerke verschiedene Spin-Up Zeiten haben. Dies bedeutet, daß das eingebaute Laufwerk des ATARI 1040 ST seine Nenndrehzahl schneller erreicht als z.B. die "alte" SF 354, so daß Programmteile, die direkt auf den Floppy-Controller zugreifen, bei den verschiedenen Laufwerken unterschiedlich funktionierten (mal ja, mal nicht). Dieser Fehler trat in der Testphase auf, ist aber jetzt natürlich behoben. Sollten Sie jedoch irgendwelche exotischen Laufwerke angeschlossen haben (alte 5 1/4 Zoll) und die Menüpunkte READ with SYNCS oder XFORMAT nicht funktionieren, so lassen Sie bitte einfach den Laufwerksmotor durch frühzeitiges Verlassen des READ-with-SYNCs-Menüs laufen und rufen die gewünschte Funktion mit laufendem Motor noch einmal auf.

*Track with Syncs*

Zur praktischen Erprobung des TRACK/SYNCS Untermenues nehmen Sie bitte eine normal formatierte Diskette und wählen diesen Menuepunkt an. Dann lesen Sie bitte den Track 79 durch Anwahl von READ with SYNCS in den Speicher und schauen sich die ersten Byte an.

Als Trackvorspann sollte nun ca 60 mal $4E am Trackanfang stehen, es können jedoch auch andere Werte erscheinen (z.B. $E4, $9C, $27). Dieses Phänomen wird vom Controller verursacht, da der Lesevorgang am Trackanfang nicht synchronisiert

wird, so daß z.B. das erste Bit von $4E überlesen wird und der Controller von dieser Stelle an die nächsten 8 Datenbit als erstes Byte liest.

im Beispiel:

```
4    E    4    E    4    ..--> 9    C    9    C    9

0100 1110  0100 1110 0100 1..--> 1001 1100 1001 1100 1001
```

Nach dem Trackvorspann der ca 60 Bytes umfasst folgen 12 Nullbytes ($00) wobei das erste und letzte Nullbyte auch angeschnitten sein kann und andere Werte aufweisen kann. Im Anschluß hieran erscheinen nun die ersten Synchronisationsbytes, nämlich drei mal $A1 wobei das erste $A1 meist nicht korrekt gelesen wird. Die drei $A1 Byte schalten den Hardware-Checksum-Ermittler ein, d.h. über die nun folgenden Datenbytes ermittelt der Diskcontroller die Checksumme. Als nächstes Byte sehen Sie nun $FE welches die folgenden sechs Datenbytes als Adressfeld identifiziert. Der Inhalt dieser Bytes lautet ($4F, $00, $01, $02, $70, $1D), $4F: Track 79, $00: auf Seite 0, $01: Sektor 1, $02: der folgende Datensektor besteht aus 512 Bytes,
$70 $1D: Checksumme über das Adressfeld. Nun folgen wieder 22 mal $4E und 12 mal $00 gefolgt von 3 mal $A1. Das nächste Byte ($FB) kündigt die folgenden 512 Datenbyte (alles $E5 auf einer frisch formatierten Diskette) gefolgt von 2 Checksumbytes ($C4 $0B) an. Ans Ende des Datensektors wurde 40 mal $4E geschrieben und danach wiederholen sich die einzelnen Synchronisationsbereiche ($00, $4E, $A1 ect.) für die noch folgenden acht Adress- und Datenfelder diese Tracks.

Verlassen Sie doch nun einmal das Track/Sync Menue und wählen aus dem Format-Menue den Unterpunkt GAP an. Nun erhöhen Sie bitte mit den Cursortasten den Wert für GAP2 von 12 auf 15, wählen danach XFORMAT und bestätigen die Abfrage mit y. Wenn Sie sich nun den so formatierten Track Nr. 79 nocheinmal mittels TRACK/SYNC-Menue anschauen sehen Sie nunmehr jeweils 15 mal $00 zwischen den einzelnen Adress- und Datenfeldern.

## 7.4 Das Assemblieren mit verschiedenen Assemblern

*Digital Research:*

1. Assemblieren mittels: as68.ttp -l -u editor.s
2. Linken mit: link68.ttp [u] edit.68k=edit.o
3. Ladbar machen mit: relmod.ttp edit.68k edit.tos

Das File edit.tos ist durch anklicken ladbar.

*GST-Assembler:*

1) An den Anfang des Files "edit.s" muß " opt abs " eingefügt und alle Speicherplatzdirektiven wie " text, bss, data " müssen durch " section " angeführt werden. Aus " text " wird somit " section text ". Das Programmfile nach dem Ändern mit dem Namen "edit.gst" abspeichern.

2) Erstellen eines Linkfiles mit Namen "asl.lnk", welches nur die eine Zeile " input * " enthält.

3) Assemblieren von "edit.gst" mit asm.prg edit.gst -errors

4) Linken mit link.prg edit -with asl.lnk -prog edit.tos

Das File edit.tos ist wiederum startbar.

*Metacomco-Assembler*

Das Programmfile "edit.s" kann ohne Änderung übernommen werden.

1) Erstellen eines Linkfiles "asl.lnk" welches als einzige Zeile " input *" enthält.

2) Assemblieren mit assem.ttp edit.s to edit.bin

3) Linken mit link.ttp edit.bin -with asl.

Das so entstandene Programmfile "edit.prg" ist startbar.

# 8. Maschinen-Hilfsprogramme für BASIC

Das BASIC des ATARI ST ist mit recht vielen Funktionen ausgestattet und dabei noch sehr schnell. Dennoch tauchen immer wieder Probleme auf, die in BASIC nicht oder nur sehr schwer gelöst werden können. Auch Zeitprobleme entstehen oft da, wo große Datenmengen zu verwalten sind.

Abhilfe schafft hier meist ein mehr oder weniger kleines Maschinen-Programm, welches in das BASIC-Programm eingebunden werden kann. Ein solches Unterprogramm kann leicht in ein Integer-Feld (z.B. A%(n) ) abgelegt und dort aufgerufen werden. Dabei sind einige Dinge zu beachten, welche wir nun betrachten wollen.

## 8.1 Aufruf und Parameterübergabe

Es gibt in nahezu jedem BASIC-Dialekt zwei Befehle, die das Zusammenspiel zwischen BASIC und Assembler ermöglichen. Diese beiden Befehle lauten USR und CALL.

Leider ist beim ATARI-BASIC die USR-Funktion nicht implementiert. Eine ältere Version gab dies sogar mit der Meldung 'Function not yet implemented' zu. Wir müssen uns daher mit der anderen Funktion näher beschäftigen.

CALL ruft, wie der Name schon sagt, ein Maschinen-Programm auf. Dabei werden folgendermaßen die Parameter angegeben:

```
CALL A (P1,P2,P3)
```

A bedeutet dabei die Adresse des Maschinenprogramms. P1, P2 und P3 sind Parameter, die an das Programm übergeben werden. Die Anzahl dieser Parameter ist beliebig, also zwischen 0 und sehr vielen.

Wird ein direkter Wert (z.B. 1) angegeben, so wird dieser so übernommen. Setzt man eine Variable ein, so wird deren Inhalt angenommen. Dabei ist zu beachten, daß die Werte in einem Langwort übergeben werden, so daß nur Werte zwischen minus und plus 2 Milliarden zulässig sind. Fließkommazahlen können nicht verwendet werden.

Bei der Übergabe eines Stringvariablen (z.B. A$) wird die Adresse des Strings im Speicher übernommen. Dadurch spart man sich den Umweg über die VARPTR-Funktion. Diese Funktion braucht man dennoch zur Errechnung der Anfangsadresse des Programms.

Das Maschinen-Programm findet nun auf dem Stack folgende Parameter:

Zuerst einmal die Rücksprung-Adresse, zu der die Kontrolle des Prozessors beim RTS-Befehl zurückkehrt. Dieser Wert ist meist uninteressant, er darf jedoch auf keinen Fall verändert werden!

Danach folgt ein Wort, welches die Anzahl der übergebenen Parameter enthält. Man kommt an dieses Wort einfach durch den Befehl MOVE.W 4(SP),D0 heran, wobei die Anzahl nun im Register D0 liegt.

Das nun folgende Langwort enthält einen Zeiger auf die Parameter-Liste selbst. Dieser Zeiger kann z.B. durch MOVE.L 6(SP),A0 in das Adreß-Register A0 geladen werden.

Die Parameter-Liste enthält nun in der im BASIC-Aufruf gegebenen Reihenfolge die Langworte der Parameter. Diese Werte können nun im Maschinen-Programm weiterverarbeitet werden.

Bei den Experimenten mit dieser Funktion stieß ich auf einen merkwürdigen Effekt. Einige Programme liefen einwandfrei, andere, zum Teil einfachere, ließen den Rechner abstürzen. Nach mehrmaligen Haareraufen und Aschenbecherleeren kam ich schließlich auf des Rätsels Lösung: In dem Maschinen-Programm darf nicht das Adreß-Register A6 verwendet bzw. ver-

ändert werden! Nach der Änderung aller A6 im Programm in A4 liefen die Programme sofort einwandfrei...

## 8.2 Einige Beispielprogramme

In den nun folgenden Kapiteln sollen einige Unter-Routinen für BASIC-Programme vorgestellt werden. Dabei werden einige Problemlösungen gegeben, die Sie hoffentlich auch für Ihre eigenen Programme verwenden können. Außerdem können Sie anhand der Beispielprogramme sehen, wie die Verwendung und der Austausch verschiedener Parameter vor sich gehen kann. Damit können Sie dann auch andere Maschinen-Programme schreiben, die Ihr BASIC-Programm wesentlich verbessern und beschleunigen können.

Die Beispiele sind immer jeweils in Assembler- und in BASIC-Listings gegeben. Das BASIC-Programm enthält dabei auch je einen Lader, mit dem das Maschinenprogramm generiert wird. Sie können natürlich auch so vorgehen, daß Sie die Daten des Maschinen-Programmes in einer Datei auf Diskette ablegen und mit dem Befehl BLOAD "Filename",A in das Feld einlesen. Dadurch wird zwar ein zusätzlicher Diskettenzugriff nötig, das BASIC-Programm wird jedoch kürzer und übersichtlicher.

### 8.2.1 Schnittstelle BASIC/TOS

Das Betriebssystem des ATARI ST bietet viele Funktionen, an die man jedoch aus einem BASIC-Programm heraus nicht herankommt.

Mit Hilfe eines Maschinenprogramms kann dieses Problem jedoch leicht gelöst werden. Ein solches Programm muß die Möglichkeit bieten, eine beliebige Anzahl von Parametern vom aufrufenden BASIC-Programm zu übernehmen und auf dem Stack dem Betriebssystem zu übergeben.

Das nun folgende Programm besitzt diese Möglichkeit. Damit stellt es eine universelle Schnittstelle zwischen BASIC und dem

Betriebssystem dar. Der Aufruf des Programms erfolgt mit einer beliebigen Anzahl von Parametern. Dabei ist jedoch zu beachten, daß nur Daten mit Wortbreite (16 Bit) übernommen werden. Ein Langwort muß daher in zwei Teilen übergeben werden.

Der letzte Parameter, der im CALL-Befehl angegeben wird, hat eine besondere Bedeutung. Da einige Funktionen des GEMDOS einen Wert zurückgeben, wird dieser Wert in die zuletzt angegebene Adresse übergeben. Im Beispielprogramm (BASIC) wird die Funktion CONIN demonstriert, die auf die Betätigung einer Taste wartet und den Wert dieser Taste zurückgibt.

Doch sehen wir uns zuerst das Maschinenprogramm selbst an. Die übergebenen Parameter werden in einer Schleife auf den Stack übertragen und dann mit einem TRAP-Befehl an das Betriebssystem übergeben. Der Rückgabewert, der im allgemeinen im Datenregister D0 zurückkommt, wird nun an die Adresse des letzten Parameters geschrieben.

```
          ;** BASIC-TOS-Schnittstelle 6/86 S.D. **
          ;** Aufruf mit CALL ADR (Parameter-Liste,x) **
          ;** mit x als Adresse des Rückgabewertes D0 **

run:
          move      4(sp),d0          ;Anzahl der Parameter
          move.l    6(sp),a5          ;Zeiger auf Parameterblock
          subq      #2,d0             ;Parameteranzahl korrigieren
          move.l    sp,a4             ;alten Stackpointer retten

loop:
          move.l    (a5)+,d1          ;Parameter holen
          move      d1,-(sp)          ;und auf den Stack damit
          dbra      d0,loop           ;weitermachen

          trap      #1                ;TOS aufrufen
          move.l    (a5),a5           ;Rückgabeadresse ermitteln
          move.l    d0,(a5)           ;D0 zurückgeben
          move.l    a4,sp             ;Stack reparieren
          rts                         ;Ende!
```

Wie Sie sehen, ist das Programm ausgesprochen einfach. Wir wollen deshalb gleich zu dem BASIC-Programm übergehen, welches das Maschinenprogramm erstellt und danach einen Probelauf macht. Die dabei eingesetzte Funktion ist, wie schon erwähnt, die CONIN-Funktion, die auf einen Tastendruck wartet und den Wert der gedrückten Taste in D0 zurückgibt. Dabei bedeutet das niederwertige Wort von D0 den ASCII-Wert der Taste, das höherwertige Wort enhält den rechnerinternen Scan-Code. Beide Werte erhält man im Langwort D0, welches in die String-Variable B$ geschrieben wird.

```
10 '** BASIC-TOS-Schnittstelle S.D. **
20 defdbl s
30 dim a%(200)
40 a=varptr(a%(0))
50 s=0
60 for i=0 to 14 :read a%(i)
70 s=s+a%(i) :next i
80 if s<> 174830 then ?"Fehler!":stop
100 b$=space$(10)
110 b=varptr(b$) :'Adresse für die Rückgabe
120 call a (1,b) :'Aufruf der Routine
130 ?peek(b),b$ :'Ausgabe des Ergebnisses

1000 '** Daten für BASTOS **
1010 data &H302F,4,&H2A6F,6,&H5540,&H284F,&H221D,&H3F01
1020 data &H51C8,&HFFFA,&H4E41,&H2A55,&H2A80,&H2E4C,&H4E75
```

## 8.2.2 Directory auslesen

Ein ärgerlicher Mangel des ATARI-BASIC ist die fehlende Möglichkeit, das Inhaltsverzeichnis einer Diskette auszulesen. Man kann zwar mit dem Befehl DIR das Directory auf dem Bildschirm ausgeben lassen, doch was nützt das schon?

Will man nun aber in einem Programm die Informationen, die das Directory beinhaltet, verwenden, so muß man wieder einmal auf ein Maschinen-Programm zurückgreifen. Ein solches Programm finden Sie in diesem Kapitel. Es kann sogar außer dem normalen Zugriff auf die Dateinamen noch alle anderen Informationen liefern, die im Inhaltsverzeichnis stehen (siehe Kapitel 6.3). Zusätzlich liefert es noch die Angaben über die gesamte und die verbliebene Kapazität der aktuellen Diskette.

Doch betrachten wir erst einmal das Maschinen-Programm selbst.

```
    ;** Directory für BASIC S.D. **

run:
    bra       sfirst            ;Einsprung 1

snext:                          ;Einsprung 2
    move      #$4f,-(sp)
    trap      #1                ;snext-Funktion
    addq.l    #2,sp
    tst       d0
    bne       warnix            ;keine weiteren Einträge
    rts

sfirst:
    cmp       #3,4(sp)
    bne       quit              ;keine 3 Parameter!

    move.l    6(sp),a5          ;Zeiger auf Parameterblock
    lea       puffer(pc),a4
    move.l    8(a5),(a4)        ;Pufferadresse retten

    move.l    8(a5),-(sp)       ;Pufferadresse
    move      #$1a,-(sp)
    trap      #1                ;SETDTA-Funktion
    addq.l    #6,sp

    move      6(a5),-(sp)       ;Attribut
```

```
        move.l  (a5),-(sp)          ;Filename
        move    #$4e,-(sp)
        trap    #1                  ;SFIRST-Funktion
        addq.l  #8,sp

        tst     d0
        bne     warnix

quit:
        rts                         ;=> BASIC

warnix:
        move.l  puffer(pc),a4
        clr     -(sp)               ;aktuelles Laufwerk
        move.l  a4,-(sp)            ;Puffer-Adresse
        move    #$36,-(sp)
        trap    #1                  ;GET-FREE-SPACE-Funktion
        addq.l  #8,sp

        move.l  #'Frei',30(a4)      ;Kein Filename!
        rts                         ;=> BASIC

puffer: dc.l  0
```

Das erste, was an dem Programm auffällt, sind die zwei unterschiedlichen Einsprung-Punkte. Dies ist deshalb so, weil das Programm eigentlich aus 2 eigenen Programmen besteht.

Der erste Teil ist die 'SEARCH_FIRST'-Funktion. Dieser Funktion des GEMDOS müssen einige Parameter übergeben werden, wie Such-Name, File-Attribut und Pufferadresse. Die Funktion des anderen Programmteils, 'SEARCH_NEXT', benötigt dagegen keine Parameter, da die Einstellungen des letzten 'SEARCH_FIRST'-Aufrufes weiterverwendet werden.

Für das aufrufende BASIC-Programm heißt dies, daß es erst das Programm am Anfang aufrufen muß und dann mit der Adresse+4 weiterverfahren muß. Außerdem muß nur einmal die Parameterübergabe vorgenommen werden.

Findet die SEARCH_FIRST- oder die SEARCH_NEXT-Funktion kein weiteres File, welches den Suchkriterien entspricht, so wird eine weitere Funktion aufgerufen. Diese Funktion gibt einen Parameterblock zurück, in dem die Informationen über Gesamtgröße und Restkapazität der aktuellen Diskette enthalten sind. Diese Informationen erhält das BASIC-Programm in dem selben Parameter-Block zurück wie auch die Directory-Einträge. Es kann sie jedoch daran erkennen, daß als Programmname 'Frei' übergeben wird.

Der erste Aufruf des Programms lautet

```
CALL A (F$,A,P$)
```

Die Bedeutung der Parameter:

| | |
|---|---|
| A | gibt die Anfangsadresse des Maschinen-Programms an. |
| F$ | ist ein String, in dem der Pfadname der zu suchenden Dateien enthalten ist (z. B. B:*.BAS). Der String muß mit einem Nullbyte abgeschlossen sein! |
| A | ist das Attribut, welches die zu suchenden Dateien haben sollen. Eine Null sucht nach allen normalen Dateien. |
| P$ | bezeichnet einen String, der als Puffer für die vom Maschinen-Programm zurückgegebenen Daten dient. Die Aufteilung des Puffers entnehmen Sie bitte dem Kapitel 6.3. |

Hier nun ein BASIC-Programm, welches das Maschinen-Programm generiert und gleich ein Anwendungsbeispiel darstellt. Es werden dabei alle Dateien der im aktuellen Laufwerk liegenden Diskette mit ihrer Länge ausgegeben. Im Anschluß daran erhält man die noch auf der Diskette vorhandenen Kapazität in Bytes.

```
10 '** Directory lesen S.D. **
20 defdbl s
30 dim a%(200)
40 d=varptr(a%(0)) :'1. Einsprung für SEARCH_FIRST
50 s=0
60 for i=0 to 52 :read a%(i)
70 s=s+a%(i) :next i
80 if s<> 610895 then ?"Fehler!":stop

130 d1=d+4 :'2. Einsprung für SEARCH_NEXT
150 f$="\*.*"+chr$(0) :'Suchstring
160 p$=space$(50) :'Puffer löschen
170 call d (f$,0,p$) :'SEARCH_FIRST
180 goto lop1
190 loop:
200 call d1 :'SEARCH_NEXT
210 lop1:
220 if mid$(p$,31,4) = "Frei" then 260 :'Ende
230 i=27: gosub calc :'Länge berechnen
240 ?mid$(p$,31,14),l :'Name und Länge ausgeben
250 goto loop

260 i=1: gosub calc :'freie KBytes berechnen
270 ?"** Freie Bytes : ";l*1024 :'und in Bytes ausgeben
280 end :'Programm-Ende

290 calc:
300 l=asc(mid$(p$,i+3,1))+&H100*asc(mid$(p$,i+2,1))
310 l=l+&H10000*asc(mid$(p$,i+1,1))
320 return

1000 '** Daten für BASDIR **
1010 data &H6000,&H12,&H3F3C,&H4F,&H4E41,&H548F,&H4A40
1020 data &H6600,&H3C,&H4E75,&HC6F,3,4,&H6600,&H2E,&H2A6F
1030 data 6,&H49FA,&H42,&H28AD,8,&H2F2D,8,&H3F3C
1040 data &H1A,&H4E41,&H5C8F,&H3F2D,6,&H2F15,&H3F3C,&H4E
1050 data &H4E41,&H508F,&H4A40,&H6600,4,&H4E75,&H287A,&H18
1060 data &H4267,&H2F0C,&H3F3C,&H36,&H4E41,&H508F,&H297C
1070 data &H4672,&H6569,&H1E,&H4E75,0,0
```

### 8.2.3 Sektoren lesen/schreiben

Die Daten auf einer Diskette sind, wie bereits besprochen, in Sektoren abgelegt. An diese Sektoren kommt man normalerweise nicht direkt heran, das Betriebssystem läd nur die Sektoren, auf denen die gewählte Datei steht.

Will man nun auf bestimmte Sektoren zugreifen, so muß man wieder einmal ein Maschinenprogramm bemühen, welches einen einzelnen Sektor lesen bzw. schreiben kann. Ein solches Programm wird nun vorgestellt.

Dem Programm werden 3 Parameter übergeben: die logische Sektornummer, eine Lese- bzw. Schreib-Anweisung und die Adresse des zu verwendenden Puffers.

Die logische Sektornummer kann von 0 bis zum maximalen Wert sein. Dieses Maximum hängt von dem verwendeten Disketten-format ab.

Die Lese/Schreibe-Anweisung kann folgende Werte annehmen:

0 - Sektor lesen
1 - Sektor schreiben
2 - Sektor lesen, Diskettenwechsel ignorieren
3 - Sektor schreiben, Diskettenwechsel ignorieren

Wird das Kommando 0 oder 1 verwendet, so greift das Programm nur auf die Diskette zu, die momentan im Laufwerk liegt. Ein Diskettenwechsel bewirkt, daß nicht zugegriffen wird.

Das Maschinenprogramm sieht folgendermaßen aus:

```
;** Sektor lesen S.D. **
;** CALL A (Sektor,rw (2=read,1=write),Puffer) **

run:
```

```
        cmp      #3,4(sp)        ;3 Parameter?
        bne      quit            ;nein => Abbruch

        move.l   6(sp),a5        ;Zeiger auf Parameter

        clr      -(sp)           ;Laufwerk A
        move     2(a5),-(sp)
        move     #1,-(sp)        ;1 Sektor
        move.l   8(a5),-(sp)     ;Puffer
        move     6(a5),-(sp)     ;Read/Write
        move     #4,-(sp)        ;RWABS-Funktion
        trap     #13             ;BIOS-Aufruf
        add.l    #14,sp

quit:
        rts                      ;=> BASIC
```

Das Programm greift nur auf Laufwerk A zu. Sollte dies ebenfalls variabel sein, so können Sie das Programm auch für 4 Parameter umschreiben.

Hier nun das entsprechende BASIC-Programm, welches das Maschinen-Programm erstellt und gleich eine kleine Demonstration darstellt:

```
10 '** Durchsuchung eines Integer-Feldes S.D. **
30 dim a%(100),f%(300)
40 a=varptr(a%(0))
50 defdbl s
60 s=0
70 for i=0 to 22: read a%(i) :'Programm einlesen
80 s=s+a%(i) :next i
90 if s<> 165974 then ?"Fehler!" :stop
100 f=varptr(f%(0))

200 input "Sektor, rw : ";s%,r%
210 call a (s%,r%,f) :'Aufruf des Programms
220 for i=0 to 255
```

```
230 if (i mod 16)=0 then ?
240 ?mki$(f%(i)); :'ASCII-Ausgabe des Sektors
250 next i :?

1000 '** Daten für Maschinenprogramm **
1010 data &HC6F,3,4,&H6600,&H24,&H2A6F,6,&H4267
1020 data &H3F2D,2,&H3F3C,1,&H2F2D,8,&H3F2D,6
1030 data &H3F3C,4,&H4E4D,&HDFFC,0,&HE,&H4E75
```

### 8.2.4 Beliebige Diskettenformatierung

Wie wir bereits im 6. Kapitel gesehen haben, lassen sich die 3½-Zoll-Disketten in verschiedenen Formaten verwenden. Dabei sind die Anzahl der verwendeten Seiten, der Tracks und der Sektoren pro Track variabel.

Um nun aus einem BASIC-Programm heraus eine Diskette zu formatieren, ist man auf zusätzliche Unterstützung von einem Maschinen-Unterprogramm angewiesen, da ein entsprechender BASIC-Befehl dafür im Befehlsvorrat fehlt. Außerdem könnten auch mit dem Umweg über das Desktop nur zwei verschiedene Formate verwendet werden. Abhilfe schafft hier ein Maschinenprogramm, welches vom BASIC aus aufgerufen und mit einigen Parametern versorgt werden kann. Diese Parameter ergeben dann das Format, in dem die angegebene Diskette initialisiert wird.

Das Programm selbst erinnert grob an jenes, welches wir im 6. Kapitel kennengelernt haben. Bei der näheren Betrachtung fallen jedoch einige gravierende Unterschiede auf.

Zum einen fehlt das Menü und die damit verbundene Parameterberechnung. Alle wichtigen Einstellungen werden nämlich direkt vom aufrufenden BASIC-Programm übergeben.

Zum anderen sind die Adressierungen der Variablen anders geartet. Dies ist aus dem Grunde so kompliziert, da das Programm mit einem BASIC-Lader in einen Speicherbereich gelesen wird, der dem Programm selbst unbekannt ist. Alle Adressierun-

gen müssen somit relativ sein, absolute Adressen gibt es hier keine.

Aufgerufen wird das Maschinenprogramm durch einen CALL-Befehl folgenden Aufbaus:

```
CALL A (S,T,SPT,LW)
```

Die verwendeten Variablen haben folgende Bedeutung:

A ist die Speicheradresse, an die das Maschinenprogramm gelegt wurde. Im vorliegenden Beispiel ist dies die mit der VARPTR-Funktion ermittelte Adresse des Integer-Feldes A%.

S steht für die Seitenanzahl, die auf der Diskette formatiert werden soll. Dabei ist die Anzahl-1 zu übergeben (einseitig: S=0, doppelseitig: S=1).

T gibt die Anzahl der Tracks an. Normalerweise enthält eine Diskette 80 Tracks, physikalisch sind jedoch bis zu 82 (manchmal sogar 83) Tracks formatierbar.

SPT sind Sektoren pro Track. Hier steht im Normalfall eine 9, es sind jedoch 1 bis 10 Sektoren pro Track formatierbar.

LW heißt Laufwerk. Mit dieser Variablen wird das zu formatierende Laufwerk bestimmt, wobei eine 0 für Disk A und eine 1 für Disk B eingesetzt werden muß. Bitte versuchen Sie nicht, durch Einsetzen einer 3 für Laufwerk C Ihre RAM-Disk zu formatieren; dies spricht beide Laufwerke (A und B) gleichzeitig an...

Hier nun das Maschinen-Programm, welches die Parameter vom BASIC annimmt, auswertet und die Diskette formatiert:

```
;** BASIC-Unterprogramm: Formatierungs-Routine S.D. **

run:
        move     4(sp),d0
        cmp      #4,d0                   ;4 Parameter?
        bne      quit                    ;nein => Abbruch
        move.l   6(sp),a5                ;Zeiger auf Parameterblock

        lea      seiten(pc),a4
        move.l   (a5)+,d1
        move     d1,(a4)                 ;Seiten
        move.l   (a5)+,d1
        move     d1,2(a4)                ;Tracks
        move.l   (a5)+,d1
        move     d1,4(a4)                ;Sektoren pro Track
        move.l   (a5)+,d1
        move     d1,6(a4)                ;Laufwerks-Nummer

        move     tracks(pc),8(a4)
        subq     #1,8(a4)

floop:
        move     seiten(pc),10(a4)       ;Seite bestimmen
floop1:
        bsr      fmttr                   ;format Track
        bne      quit
        sub      #1,10(a4)               ;Seite -1
        bpl      floop1
        sub      #1,8(a4)
        bpl      floop                   ;nächster Track

setboot:
        clr      -(sp)                   ;Execute-Flag
        moveq    #2,d0
        or       seiten(pc),d0
        move     d0,-(sp)                ;Disktyp, Seiten
        move.l   #$1000000,-(sp)         ;Seriennr. erstellen
        pea      12(a4)                  ;Puffer-Adresse
```

```
        move    #$12,-(sp)
        trap    #14                 ;Boot-Sektor erstellen
        add.l   #14,sp

        lea     12(a4),a0
        clr.l   d0
        cmp     #9,4(a4)            ;9 Sektoren pro Track?
        beq     sok                 ;ja
        move.b  #10,24(a0,d0)       ;10 SPT einsetzen
        move    tracks(pc),d1
        tst     (a4)                ;1 Seite?
        beq     sd11                ;ja
        lsl     #1,d1               ;sonst doppelter Zuwachs
sd11:
        bsr     addsec              ;SEC + Anzahl der Tracks

sok:
        cmp     #80,2(a4)           ;80 Tracks?
        beq     trok
        move    #18,d1
        tst     (a4)                ;1 Seite?
        beq     sd12                ;ja
        lsl     #1,d1               ;sonst doppelter Zuwachs
sd12:
        bsr     addsec              ;SEC + 2*9 oder 4*9

trok:
        move    #1,-(sp)            ;1 Sektor
        clr.l   -(sp)               ;Seite 0, Track 0
        move    #1,-(sp)            ;Sektor 1
        move    drive(pc),-(sp)     ;Laufwerk
        clr.l   -(sp)
        pea     12(a4)              ;Puffer
        move    #9,-(sp)
        trap    #14                 ;flopwr
        add.l   #20,sp

quit: rts                           ;zurück zum BASIC

addsec:                             ;SEC = SEC + D1
```

```
        move.b   20(a0,d0),d2       ;HI
        lsl      #8,d2
        move.b   19(a0,d0),d2       ;LO
        add      d1,d2
        move.b   d2,19(a0,d0)       ;set LO
        lsr      #8,d2
        move.b   d2,20(a0,d0)       ;set HI
        rts

fmttr:                              ;einen Track formatieren
        clr      -(sp)              ;Virgin-Daten
        move.l   #$87654321,-(sp)   ;Magic-Zahl
        move     #1,-(sp)           ;interleave
        move     seite(pc),-(sp)    ;Seite
        move     tracks1(pc),-(sp)  ;Track
        move     secptr(pc),-(sp)   ;Sektoren/Track
        move     drive(pc),-(sp)    ;Laufwerk
        clr.l    -(sp)
        pea      12(a4)
        move     #10,-(sp)
        trap     #14                ;flopfmt
        add.l    #26,sp
        tst      d0                 ;Test auf Error
        rts

seiten:  dc.w 1
tracks:  dc.w 80
secptr:  dc.w 9
drive:   dc.w 0
tracks1: dc.w 80
seite:   dc.w 0
puffer:  dc.b $200
```

Das Programm ist voll verschiebbar, d.h. es läuft so an jeder beliebigen Speicheradresse. Die einzelnen Komponenten des Programms sind bereits in diesem Buch erklärt, so daß es eigentlich selbsterklärend ist.

Nun folgt ein BASIC-Programm, welches die Formatierungsroutine aufruft. Gleichzeitig ist ein Lader enthalten, welches das Maschinen-Programm aus DATA-Zeilen generiert. Selbstverständlich kann es auch von der Diskette geladen werden.

```
10 '** Formatierung einer Diskette **
15 'fullw 2
17 defdbl s
20 dim a%(400)
30 a=varptr(a%(0))
35 s = 0
40 for i=0 to 144: read a%(i)
45 s =s +a%(i) : next i
46 if s <> 1033402 then ?"Fehler!":stop
50 print "Seiten, Tracks, Sektoren/Track, Laufwerk "
60 input s,t,spt,lw
80 call a (s,t,spt,lw)
90 '** Daten für BFORMAT.obj **
100 data &H302F,&H0004,&H0C40,&H0004,&H6600,&H00CC,&H2A6F,&H0006
110 data &H49FA,&H0110,&H221D,&H3881,&H221D,&H3941,&H0002,&H221D
120 data &H3941,&H0004,&H221D,&H3941,&H0006,&H397A,&H00F8,&H0008
130 data &H536C,&H0008,&H397A,&H00EC,&H000A,&H6100,&H00B4,&H6600
140 data &H0096,&H046C,&H0001,&H000A,&H6A00,&HFFF0,&H046C,&H0001
150 data &H0008,&H6A00,&HFFE0,&H4267,&H7002,&H807A,&H00C6,&H3F00
160 data &H2F3C,&H0100,&H0000,&H486C,&H000C,&H3F3C,&H0012,&H4E4E
170 data &HDFFC,&H0000,&H000E,&H41EC,&H000C,&H4280,&H0C6C,&H0009
180 data &H0004,&H6700,&H0018,&H11BC,&H000A,&H0818,&H323A,&H0096
190 data &H4A54,&H6700,&H0004,&HE349,&H6100,&H003E,&H0C6C,&H0050
200 data &H0002,&H6700,&H0012,&H323C,&H0012,&H4A54,&H6700,&H0004
210 data &HE349,&H6100,&H0024,&H3F3C,&H0001,&H42A7,&H3F3C,&H0001
220 data &H3F3A,&H0066,&H42A7,&H486C,&H000C,&H3F3C,&H0009,&H4E4E
230 data &HDFFC,&H0000,&H0014,&H4E75,&H1430,&H0814,&HE14A,&H1430
240 data &H0813,&HD441,&H1182,&H0813,&HE04A,&H1182,&H0814,&H4E75
250 data &H4267,&H2F3C,&H8765,&H4321,&H3F3C,&H0001,&H3F3A,&H002E
260 data &H3F3A,&H0028,&H3F3A,&H0020,&H3F3A,&H001E,&H42A7,&H486C
270 data &H000C,&H3F3C,&H000A,&H4E4E,&HDFFC,&H0000,&H001A,&H4A40
280 data &H4E75
```

### 8.2.5 Daten suchen

Eine recht häufige Anwendung eines Maschinen-Unterprogrmms ist das Durchsuchen von Listen. Eine solche Suche kann bei größeren Listen so lange dauern, daß ein BASIC-Programm nicht mehr sinnvoll ist. Man denke sich nur eine Datenbank, die zum Suchen einer Telefonnummer etliche Minuten braucht...

Ein Maschinenprogramm, welches diese Aufgabe übernimmt, ist recht leicht zu schreiben. Es besteht nur aus drei Teilen:

1. Parameterübernahme vom BASIC
2. eine Suchschleife
3. Ergebnisrückgabe an das BASIC-Programm

Hier nun ein solches Programm:

```
        ;** Integer-Feld-Durchsuchung S.D. **
        ;** CALL A (Feldanfang,Anzahl,Suchwort) **

run:
        cmp      #3,4(sp)        ;3 Parameter?
        bne      quit            ;nein => Exit

        move.l   6(sp),a5        ;Zeiger auf Parameter

lop1:
        move.l   (a5),a4         ;Zeiger auf Parameterfeld
        tst      (a4)+           ;auf f%(1) stellen
        move.l   4(a5),d1        ;Anzahl der Daten
        move.l   8(a5),d2        ;Suchwort
        moveq    #1,d3           ;Index=1
loop:
        cmp      (a4)+,d2        ;Vergleich
        beq      ok1             ;gefunden
        addq     #1,d3           ;Index+1
        cmp      d3,d1           ;Ende?
```

```
        bne     loop            ;nein

        move    #-1,d3          ;nicht gefunden!
ok1:
        move.l  (a5),a5         ;Adresse für Rückgabe
        move    d3,(a5)         ;Index zurückgeben

quit:
        rts                     ;=> BASIC
```

Dieses kleine Programm erledigt die gesamte Suche in Bruchteilen von Sekunden und gibt die Nummer des gesuchten Eintrages in das erste Element der Liste zurück. Daher ist zu beachten, daß man nur die Elemente 1 bis n der Liste für die Daten verwendet. Wird das Element nicht gefunden, so wird als Nummer -1 übergeben.

Da es für dieses Programm viele Verwendungsmöglichkeiten gibt, ist das folgende Lade- und Beispielprogramm nur sehr einfach. Das Prinzip der Routine und ihrer Verwendung wird aber klar.

```
10 '** Durchsuchung eines Integer-Feldes S.D. **
30 dim a%(60),f$(1000)
40 a=varptr(a%(0))
50 defdbl s
60 s=0
70 for i=0 to 25 :read a%(i)
80 s=s+a%(i) :next i
90 if s<> 211865 then ?"Fehler!" :stop
130 f=varptr(f%(0))

140 for i%=1 to 8
150 read f%(i%) :'Beispielwerte einlesen
170 next i%

200 input "Suchwort : ";s%
210 call a (f,i%,s%)
```

```
220 if f%(0)=-1 then ?"** Nicht gefunden! **" :goto 200
230 ?s%;" ist der ";f%(0);". Eintrag." :goto 200

1000 '** Daten für Maschinenprogramm **
1010 data &HC6F,3,4,&H6600,&H2A,&H2A6F,6,&H2855
1020 data &H4A5C,&H222D,4,&H242D,8,&H7601,&HB45C,&H6700
1030 data &HE,&H5243,&HB243,&H6600,&HFFF4,&H363C,&HFFFF
1040 data &H2a55,&H3A83,&H4E75
1100 data 6,2,99,345,7,3,0,4
```

### 8.2.6 Daten sortieren

Die Sortierung von großen Datenmengen ist eine sehr zeitintensive Sache. Ein BASIC-Programm, welches einige 1000 Daten sortiern soll, macht dabei eine unangenehm lange Pause, die den Programm-Ablauf empfindlich stören kann. Wesentlich schneller ist dagegen ein Maschinen-Programm, welches diese Aufgabe übernimmt.

Ein solches Programm soll nun vorgestellt werden. Es ist dafür ausgelegt, ein beliebig großes Integer-Feld eines BASIC-Programmes aufsteigend zu sortieren. Dem Programm werden als Parameter die Adresse des Feldanfangs und die Anzahl der zu sortierenden Einträge übergeben. Dadurch kann auch ein Ausschnitt eines Feldes für sich sortiert werden.

Der in diesem Programm verwendete Algorithmus ist recht einfach. Er ist zwar nicht der schnellste, aber das spielt bei der sehr großen Geschwindigkeit des 68000-Prozessors keine wichtige Rolle.

Hier nun das Maschinenprogramm:

```
        ;** Integer-Feld-Sortierung S.D. **
        ;** CALL A (Feldanfang,Anzahl) **

run:
        cmp     #2,4(sp)        ;2 Parameter?
```

```
        bne         quit                ;nein => Exit

        move.l      6(sp),a5            ;Zeiger auf Parameter

lop1:
        move.l      (a5),a4             ;Zeiger auf Parameterfeld
        move.l      4(a5),d1            ;Anzahl der Daten
        clr         d3                  ;Tausch-Flag löschen
lop2:
        move        (a4),d0
        cmp         2(a4),d0            ;Vergleich
        ble         ok1                 ;OK
        move        2(a4),(a4)          ;tauschen
        move        d0,2(a4)
        st          d3                  ;Tausch-Flag setzen
ok1:
        addq.l      #2,a4               ;nächster Wert
        subq.l      #1,d1
        bne         lop2

        tst         d3                  ;fertig?
        bne         lop1                ;nein => weiter

quit:
        rts                             ;=> BASIC
```

Nun wieder ein BASIC-Programm, welches in DATA-Zeilen das Maschinen-Programm enthält und daraus in ein Feld einliest. In ein anderes Feld werden danach irgendwelche Werte eingetragen. Die Eingabe wird durch den Wert -1 beendet. Es wird nun das Maschinenprogramm aufgerufen, welches die Daten des gesamten Feldes sortiert. Danach werden die sortierten Werte wieder ausgegeben.

Diese Art der Anwendung ist natürlich nur als Beispiel gedacht. Interessant wird es erst bei großen Datenmengen, wo der Geschwindigkeitsvorsprung gegenüber einem reinen BASIC-Programm deutlich wird.

```
10 '** Sortierung eines Integer-Feldes S.D. **
20 defdbl s
30 dim a%(200)
40 a=varptr(a%(0))
50 s=0
60 for i=0 to 28 :read a%(i)
70 s=s+a%(i) :next i
80 if s<> 280743 then ?"Fehler!":stop

100 dim f%(1000) :'Datenfeld vorbereiten
110 defint i
120 a=varptr(a%(0)) :'Adresse des Maschinen-Programms
130 f=varptr(f%(1)) :'Adresse der Daten
140 for i=1 to 1000
150 input "Eintrag : ";f%(i) :'Daten eingeben
160 if f%(i)=-1 then 180 :'Ende?
170 next i :'nein, weitermachen
180 call a (f,i-2) :'sortieren
190 for j=1 to i
200 ?j;".: ";f%(j) :'und wieder ausgeben
210 next j

1000 '** Daten für BASSORT **
1010 data &HC6F,2,4,&H6600,&H30,&H2A6F,6,&H2855
1020 data &H222D,4,&H4243,&H3014,&HB06C,2,&H6F00,&HC
1030 data &H38AC,2,&H3940,2,&H50C3,&H548C,&H5381,&H6600
1040 data &HFFE6,&H4A43,&H6600,&HFFD8,&H4E75
```

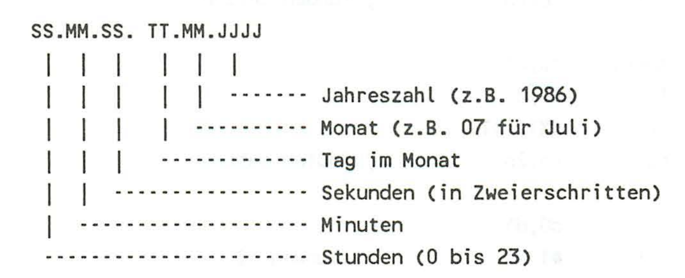
### 8.2.7 Datum und Uhrzeit formatiert auslesen

Jede Datenbank, die im täglichen Leben angewandt werden soll, muß u.a. auch das Datum und evtl. die Uhrzeit der Buchungen bzw. Änderungen mitverarbeiten können. Leider enthält das ATARI-BASIC keine entsprechende Funktion dafür, so daß wir wieder ein Maschinenprogramm brauchen.

Das nun vorgestellte Programm liest die Uhrzeit und das Datum aus dem Rechner ein und gibt beides an das aufrufende BASIC-

Programm zurück. Zusätzlich werden diese Informationen noch formatiert, damit sie direkt weiterverarbeitet werden können.

Der Aufruf geschieht einfach mit CALL A (A$), wobei in der String-Variable A$ das Ergebnis stehen wird. Das verwendete Format ist dabei folgendermaßen:




Der 3.7.1986 um 22 Uhr 16 und 30 Sekunden erscheint somit in A$ als 22.16.30. 03.07.1986.

Hier nun das Maschinen-Programm, welches die Zeit bzw. das Datum ausliest und formatiert an das BASIC-Programm zurückgibt:

```
      ;** Uhrzeit formatiert auslesen S.D. **
      ;* Aufruf mit CALL A (A$) ergibt in A$ *
      ;* SS.MM.SS. TT.MM.JJJJ., Uhrzeit und Datum *

run:
      cmp       #1,4(sp)        ;ein Parameter?
      bne       quit            ;nein => Abbruch

      move.l    6(sp),a5        ;Zeiger auf Parameterliste
      move.l    (a5),a5         ;Zeiger auf String

go:
      move      #$2c,-(sp)
      trap      #1              ;get_time-Funktion des BIOS
```

```
        addq.l  #2,sp

        and.l   #$ffff,d0       ;untere Wort ausblenden

        move    d0,d1
        lsr     #8,d1
        lsr     #3,d1           ;Stunden
        bsr     set2b           ;Stunden setzen

        move.l  d0,d1
        lsr     #5,d1
        and     #%111111,d1
        bsr     set2b           ;Minuten setzen

        move.l  d0,d1
        lsl     #1,d1           ;Sekunden *2
        and     #$3f,d1         ;und ausblenden
        bsr     set2b           ;Sekunden setzen

        move.b  #' ',(a5)+      ;Trenn-Blank setzen

        move    #$2a,-(sp)
        trap    #1              ;get_date-Funktion des BIOS
        addq.l  #2,sp

        and.l   #$ffff,d0       ;untere Wort ausblenden

        move.l  d0,d1
        and     #%11111,d1      ;Tag ausblenden
        bsr     set2b           ;Tag setzen

        move.l  d0,d1
        lsr     #5,d1
        and     #%11111,d1      ;Monat ausblenden
        bsr     set2b           ;Monat setzen

        move.l  d0,d1
        lsr     #8,d1
        lsr     #1,d1
        and     #%1111111,d1    ;Jahr ausblenden
```

```
    add      #80,d1           ;korrigieren
    move.b   #'1',(a5)+
    move.b   #'9',(a5)+       ;19oo vorbereiten
    bsr      set2b            ;Jahr setzen

quit:
    rts                       ;fertig!

set2b:                        ;D1 mit zwei Zeichen ausgeben
    divu     #10,d1
    add.l    #$300030,d1      ;ASCII-Wert korrigieren
    move.b   d1,(a5)+         ;HI-Nibble
    swap     d1
    move.b   d1,(a5)+         ;LO-Nibble
    move.b   #'.',(a5)+       ;Trenn-Punkt setzen
    rts
```

Und nun wieder das dazugehörige BASIC-Programm, welches das Maschinen-Programm erstellt und gleich eine kleine Anwendungs-Demonstration darstellt:

```
10 '** GET_TIME-Realisierung S.D. **
20 defdbl s
30 dim a%(200)
40 a=varptr(a%(0))
50 s=0
60 for i=0 to 76 :read a%(i)
70 s=s+a%(i) :next i
80 if s<> 399794 then ?"Fehler!":stop

90 t$=space$(20)
100 call a (t$)
110 ? "Heute ist der "; right$(t$,10)
120 ? "Die Zeit : "; left$(t$,8)

990 '** Daten für GETTIME **
1000 data &H0C6F,1,4,&H6600,&H7A,&H2A6F,6,&H2A55
1010 data &H3F3C,&H2C,&H4E41,&H548F,&H280,0,&HFFFF,&H3200
```

```
1020 data &HE049,&HE649,&H6100,&H5E,&H2200,&HEA49,&H241,&H3F
1030 data &H6100,&H52,&H2200,&HE349,&H241,&H3F,&H6100,&H46
1040 data &H1AFC,&H20,&H3F3C,&H2A,&H4E41,&H548F,&H280,0
1050 data &HFFFF,&H2200,&H241,&H1F,&H6100,&H2A,&H2200,&HEA49
1060 data &H0241,&H1F,&H6100,&H1E,&H2200,&HE049,&HE249,&H241
1070 data &H007F,&H641,&H50,&H1AFC,&H31,&H1AFC,&H39,&H6100
1080 data 4,&H4E75,&H82FC,&HA,&H681,&H30,&H30,&H1AC1
1090 data &H4841,&H1AC1,&H1AFC,&H2E,&H4E75
```

## 8.3 Die Programmierung des FDC von BASIC aus

Für den ATARI ST gibt es mittlerweile eine Vielzahl an Diskmonitoren. Leider werden von den uns bekannten nur die Maschinen-Routinen benutzt, die das Betriebssystem bereitstellt. Das ist für die meisten Anwendungen zwar ausreichend, wer aber genau wissen möchte, welche Informationen sonst noch auf einer Diskette verborgen sind benötigt einige Funktionen, die nur durch direkte Programmierung des Floppy-Disk-Controllers zu erreichen sind.

Gemeint sind solche Funktionen wie z.B. die ID-Felder einer Spur lesen, eine komplette Spur lesen oder eine Spur nach Belieben zu formatieren.

Unser Vorschlag: Schreiben Sie doch einfach einen Disk-Monitor, der solche Möglichkeiten bietet. In BASIC? Ja, warum denn nicht! Wenn einem die Funktionen des Floppy-Controllers zur Verfügung stehen, so ist das in BASIC durchaus möglich. Nur - über das Betriebssystem können diese nicht erreicht werden. GEMDOS, BIOS und XBIOS helfen hier also nicht weiter.

Um Abhilfe zu schaffen haben wir eine Routinensammlung erstellt die in ein BASIC-Programm eingebunden werden kann und alle Kommandos, die der FDC bereitstellt, leicht erreichbar machen.

Die Zeiten, in denen Sektoren nicht lesbar waren, weil das Betriebssystem mit 'raffinierten' ID-Feldern überlistet wurde, gehören damit der Vergangenheit an. Wenn Sie also vom Betriebsystem eine Meldung der Form 'Floppy A: antwortet

nicht ..." erhalten, können Sie die Diskette auf 'Herz und Nieren' untersuchen und feststellen weshalb etwas nicht funktionierte.

Natürlich bedarf es einiger Erfahrung, die Ergebnisse, die von den weniger gebräuchlichen Befehlen des FDC geliefert werden, zu interpretieren. In dem ausführlichen Kapitel über den Floppy-Controller WD1772 werden Sie aber alle dazu notwendigen Informationen finden, so daß einer Disketten-Analyse nichts mehr im Wege steht.

Es gibt aber noch weitere Vorteile, die die Flexibilität unserer BASIC/FDC-Schnittstelle ausmachen. Es ist jederzeit möglich die Kommandowörter zu ändern. So sind in diesen immer einige 'Option-Bits' enthalten, mit denen die Ausführung der einzelnen Kommandos beeinflußt werden kann. Ferner können - falls nicht gerade diese Kommandoworte geändert wurden - durch einen FDC-Aufruf max. alle Sektoren einer Spur, gelesen oder geschrieben werden. Es ist auch möglich z.B. nur einen Teil einer Spur zu formatieren. Das kann der Erzeugung eines besonderen Kopierschutzes dienlich sein.

Doch eines nach dem anderen. Wir benötigen zunächst einmal die FDC-Schnittstelle. Die Anwendungsmöglichkeiten werden dann später noch hinreichend erläutert.

### 8.3.1 Das BASIC/FDC-Interface-Programm

Beginnen wir mit dem Maschinenprogramm, welches unsere Schnittstelle zum FDC darstellt.

Um dieses zu erhalten gibt es drei Möglichkeiten:

1. Das Assembler-Source-Listing eingeben (die hier gedruckte Version wurde mit dem SEKA-Macro-Assembler erstellt) und assemblieren.

2. Das BASIC-Listing eingeben und mit RUN starten.

3. Die Diskette zum Buch bestellen.

Falls Sie sich für die ersten beiden Möglichkeiten entschieden haben - viel Spaß beim Tippen.

Als erstes nun das Assembler-Listing, welches Aufschluß darüber gibt, wie unsere FDC/BASIC-Schnittstelle intern arbeitet. Wer neben BASIC- auch noch Assembler-Erfahrung besitzt, findet hier alle Routinen, mit denen man dem FDC im allgemeinen zu Leibe rückt. Durch die Dokumentation dürfte es keine Schwierigkeit sein, einzelne Routinen auszukoppeln und in eigenen Programmen zu verwenden.

Die BASIC-Programmierer - denen dieses Kapitel ja schließlich gewidmet ist - mögen uns diesen Ausflug in die Welt der Maschinensprache verzeihen. Da eine direkte Programmierung des FDC aber auch von allgemeinem Interesse ist, fühlen wir uns dazu verpflichtet, auch das Assembler-
Listing zu offenbaren.

Wen also nur das fertige Programm interessiert bzw. nur über ST-BASIC verfügt, der möge bitte den folgenden Teil überspringen und mit dem BASIC-Lader 'FDCCREAT.BAS', durch welchen das Maschinenprogramm erzeugt wird, fortfahren.

```
*************************************************************
****                LISTING => FDCINTER.S                ****
*************************************************************

;*****************************************************************
;*******             FDC/BASIC-SCHNITTSTELLE             ********
;*****************************************************************
;Hardware-Register

dmamode  = $ff8606
dmascnt  = $ff8604
dmalow   = $ff860d
dmamid   = $ff860b
dmahigh  = $ff8609
```

```
giselect = $ff8800
giwrite  = $ff8802
mfp      = $fffa01

;Steuerworte für den DMA-Controller (DMA-Datenrichtung => READ)

srcmd = $80 ; Command-Register selektieren
srtrk = $82 ; Track-Register selektieren
srsec = $84 ; Sector-Register selektieren
srdat = $86 ; Data-Register selektieren
srcnt = $90 ; DMA-Sectorcount-Register selektieren

;Steuerworte für den DMA-Controller (DMA-Datenrichtung => WRITE)

swcmd = $180 ; Bedeutung wie bei => READ
swtrk = $182
swsec = $184
swdat = $186
swcnt = $190

;*****************************************************************
 even
st:
 bra.s run ; zum Programm-Start

;************************* Kommando-Worte *************************
rest: dc.w $01 ; Restore MO, 3ms Step-Rate
see:  dc.w $11 ; Seek MO, 3ms Step-Rate
stp:  dc.w $31 ; Step MO, 3ms Step-Rate, Update Trackreg.
stpi: dc.w $51 ; Step-in MO, 3ms Step-Rate, Update Trackreg.
stpo: dc.w $71 ; Step-out MO, 3ms Step-Rate, Update Trackreg.

rsec: dc.w $90 ; Read-Sector MO, multiple
wsec: dc.w $b0 ; Write-Sector MO, multiple, Write-Precompensation

radr: dc.w $c0 ; Read-Address MO,
rtrk: dc.w $e0 ; Read-Track MO,
wtrk: dc.w $f0 ; Write-Track MO, Write-Precompensation

forc: dc.w $d0 ; Force-Interrupt
```

```
;******************* Übergabe-Parameter **********************
prm: dc.w 00 ; Funktions-Nummer
     dc.w 00 ; Laufwerks-Nummer
     dc.w 00 ; Spur-Nummer
     dc.w 00 ; Sektor-Nummer
     dc.w 00 ; Anzahl zu übertragender Bytes
     dc.w 00 ; Anzahl zu lesender ID-Felder
     dc.w 00 ; FDC-Status
     dc.w 00 ; DMA-Status
     dc.w 00 ; Timeout? (1=timeout)
     dc.w 00 ; Anzahl der übertragenen Bytes
     dc.l 00 ; DMA-Start-Adresse
     dc.l 00 ; DMA-End-Adresse
     dc.l 00 ; Adresse des Spur-Puffers
     dc.l 00 ; Adresse des Sektor-Puffers
     dc.l 00 ; Adresse des ID-Puffers
     dc.l 00 ; Adresse des ID-Status-Puffers

;**************** Hier geht's richtig los ****************
run:

 tst.w 4(sp) ; wurden Parameter übergeben?
 bne exit    ; ja, zurück zum BASIC

;Da nur die Quelle PC-relativ adressiert werden darf, nehmen wir
;A3 als Programm-Zähler.

 lea st(pc),a3                       ; Programm-Start ins Adress-Reg.3
 movem.l d0-d7/a0-a6,savreg-st(a3)  ; Register retten

;******************* Set Supervisor-Mode **********************
 clr.l -(sp)                         ; Userstack => Superv.Stack
 move.w #$20,-(sp)                   ; Command => Super
 trap #1
 addq.l #6,sp                        ; Stack korrigieren
 move.l d0,savstack-st(a3)           ; alten Stackpointer retten

;*** einige Flags löschen und absolute Adresse der gewünschten ***
;*** Funktion berechnen. ***
```

```
 lea prm-st(a3),a5                   ; Zeiger auf Parameter-Block

 move.w #1,$43e                      ; Floppy-VBL sperren
 move.w #0,16(a5)                    ; Timeout-Flag löschen
 move.w #0,dma-st(a3)                ; DMA-Flag löschen
 move.w #0,vblflag-st(a3)            ; VBL-Rücksetzflag löschen

 move.w 0(a5),d0                     ; Funktionsnr. holen
 and.l #$0f,d0                       ; es gibt nur 16 Funktionen (0-15)
 lsl.l #2,d0                         ; mal 4 = functab-Offset

 lea functab-st(a3),a4               ; func-Table-Adresse
 move.l 0(a4,d0),d0                  ; Relative Start-Adresse der Routine

 jsr 0(a3,d0)                        ; +Programmstart=abs. Adr.

 tst.w vblflag-st(a3)                ; VBL einschal. (nach deselektieren)?
 beq letoff ; nein
 move.w #0,$43e                      ; einschalten

letoff:

;****************** zurück in den User-Mode *******************
 move.l savstack-st(a3),d0           ; alten Stackpointer zurückholen
 move.l d0,-(a7)                     ; alten Stackpointer übergeben
 move.w #$20,-(sp)                   ; Command => Super
 trap #1
 addq.l #6,sp                        ; Stack korrigieren

 movem.l savreg-st(a3),d0-d7/a0-a6 ; Register zurückholen

exit:
 rts                                 ; zurück zum BASIC


; Das war's! Es folgen (nur) noch die eigentlichen Routinen


;*********************** Restore FDC ****************************
```

```
restore:

 move.w #srcmd,dmamode              ; Command-Reg. selektieren
 move.w rest-st(a3),d7              ; Command => Restore
 bsr wrt1772                        ; Kommando übergeben
 bsr fdcwait                        ; Warten, bis FDC fertig
 rts

;*********************** SEEK TRACK ****************************

seek:

 move.w #srdat,dmamode              ; Daten-Reg. selektieren
 move.w 4(a5),d7                    ; Tracknr. in d7
 bsr wrt1772                        ; Tracknr. schreiben
 move.w #srcmd,dmamode              ; Command-Reg. selektieren
 move.w see-st(a3),d7               ; Command => Seek
 bsr wrt1772                        ; Command schreiben
 bsr fdcwait                        ; Warten, bis FDC fertig
 rts

;************************ Step **********************************

step:

 move.w #srcmd,dmamode              ; FDC-Commandreg. selektieren
 move.w stp-st(a3),d7               ; Command => Step
 bsr wrt1772                        ; Command schreiben
 bsr fdcwait                        ; warten, bis FDC fertig
 rts

;************************* Step in ******************************

stepin:

 move.w #srcmd,dmamode              ; FDC-Commandreg. selektieren
 move.w stpi-st(a3),d7              ; Command => Step in
 bsr wrt1772                        ; Command schreiben
 bsr fdcwait                        ; warten, bis FDC fertig
```

```
 rts

;************************* Step out ******************************

stepout:

 move.w #srcmd,dmamode              ; FDC-Commandreg. selektieren
 move.w stpo-st(a3),d7              ; Command => Step out
 bsr wrt1772                        ; Command schreiben
 bsr fdcwait                        ; warten, bis FDC fertig
 rts

;************************ Force Interrupt ***********************

Force:

 move.w forc-st(a3),d7              ; Command => Force Interrupt
 bsr wrt1772                        ; Command schreiben
 move.w #$100,d7                    ; Verzögerungsschleife
wtfrc:
 dbra d7,wtfrc
 rts

;********************** READ SECTOR(S) **************************

readsector:

 move.l 32(a5),d7                   ; DMA-Adresse auf Sektor-Buffer
 bsr setdma
 move.w #1,dma-st(a3)               ; DMA-Flag setzen
 move.w #srcnt,dmamode              ; DMA-R/W toggeln
 move.w #swcnt,dmamode
 move.w #srcnt,dmamode              ; DMA-Sectorcount selektieren
 move.w #$0c,d7                     ; mit 12 laden (entspricht 6kB)
 bsr wrt1772                        ; DMA-Scnt laden

 move.w #srsec,dmamode              ; Sector-Reg. selektieren
 move.w 6(a5),d7                    ; Sektor-Nr. in d7
 bsr wrt1772                        ; Sektor-Nr. schreiben
```

```
 move.w #srcmd,dmamode          ; Command-Reg. selektieren
 move.w rsec-st(a3),d7          ; Command => Read multiple Sectors
 bsr wrt1772                    ; Command schreiben

 bsr fdcwait                    ; warten, bis FDC fertig
 bsr readstat                   ; Status und Anzahl der Bytes lesen
 rts

;********************** Read Address ***************************

readaddress:

 move.l 40(a5),a4               ; Adresse des Statusbuffers laden
 move.l 36(a5),d7               ; DMA-Adresse auf ID-Feld-Buffer
 bsr setdma
 move.w #srcnt,dmamode          ; DMA-R/W toggeln
 move.w #swcnt,dmamode
 move.w #srcnt,dmamode          ; DMA-Sectorcount selektieren
 move.w #$01,d7                 ; mit 1 laden (entspricht 512 Byte)
 bsr wrt1772
 move.w #srcmd,dmamode          ; FDC-Commandreg. selektieren
 move.w 10(a5),d4               ; #ID-Felder in D4
 and.w #$7f,d4                  ; aber nur max. 128

idloop:
 move.w radr-st(a3),d7          ; Command => Read Address
 bsr wrt1772                    ; Command schreiben
 bsr fdcwait                    ; warten, bis FDC fertig
 move.b d0,(a4)+                ; Status in Buffer retten
 tst.w 16(a5)                   ; Timeout?
 dbne d4,idloop                 ; nein, nächstes ID-Feld lesen
 bsr readstat                   ; Status und Anzahl der Bytes lesen
 rts

;********************** READ TRACK ***************************

readtrack:

 move.l 28(a5),d7               ; DMA-Adresse auf Track-Buffer
 bsr setdma
```

```
 move.w #1,dma-st(a3)              ; DMA-Flag setzen
 move.w #srcnt,dmamode             ; DMA-R/W toggeln
 move.w #swcnt,dmamode
 move.w #srcnt,dmamode             ; DMA-Sectorcount selektieren
 move.w #$0e,d7                    ; mit 14 laden (entspricht 7kB)
 bsr wrt1772
 move.w #srcmd,dmamode             ; Command-Reg. selektieren
 move.w rtrk-st(a3),d7             ; Command => Read Track
 bsr wrt1772                       ; Command schreiben
 bsr fdcwait                       ; warten, bis FDC fertig
 bsr readstat                      ; Status und Anzahl der Bytes lesen
 rts

;********************* WRITE SECTOR(S) *************************

writesector:

 move.l 32(a5),d7                  ; DMA-Adresse auf Sektor-Buffer
 bsr setdma
 move.w #1,dma-st(a3)              ; DMA-Flag setzen
 move.w #swcnt,dmamode             ; DMA-R/W toggeln
 move.w #srcnt,dmamode
 move.w #swcnt,dmamode             ; DMA-Sectorcount selektieren
 move.w #$0c,d7                    ; mit 12 laden (entspricht 6kB)
 bsr wrt1772                       ; DMA-Scnt schreiben
 move.w #swsec,dmamode             ; Sector-Reg. selektieren
 move.w 6(a5),d7                   ; Sektornr. in d7
 bsr wrt1772                       ; Sektor-Reg. schreiben

 move.w #swcmd,dmamode             ; Command-Reg. selektieren
 move.w wsec-st(a3),d7             ; Command => Write multiple Sectors
 bsr wrt1772                       ; Command schreiben
 bsr fdcwait                       ; warten bis FDC fertig
 bsr readstat                      ; Status und Anzahl der Bytes lesen
 rts

;********************** WRITE TRACK ******************************

writetrack:
```

```
 move.l 28(a5),d7                ; DMA-Adresse auf Track-Buffer
 bsr setdma
 move.w #1,dma-st(a3)            ; DMA-Flag setzen
 move.w #swcnt,dmamode           ; DMA-R/W toggeln
 move.w #srcnt,dmamode
 move.w #swcnt,dmamode           ; DMA-Sectorcount selektieren
 move.w #$0e,d7                  ; mit 14 laden (entspricht 7kB)
 bsr wrt1772                     ; DMA-Scnt schreiben
 move.w #swcmd,dmamode           ; Command-Reg. selektieren
 move.w wtrk-st(a3),d7           ; Command => Write Track
 bsr wrt1772                     ; Command schreiben
 bsr fdcwait                     ; warten, bis FDC fertig
 bsr readstat                    ; Status und Anzahl der Bytes lesen
 rts

;*****************************************************************
;*****************************************************************

;Das waren die Routinen, über welche die Kommandos des WD1772
;angesprochen werden.
;Es folgen nun weitere Unter-Routinen, die zum Teil von den
;Haupt-Routinen und zum Teil direkt vom BASIC aus (z.B. setdrive)
;aufgerufen werden.

;**************** Sector-Register lesen *********************

rsecreg:

 move.w #srsec,dmamode           ; Sektor-Reg. selektieren
 bsr read1772                    ; und lesen
 and.w #$ff,d0                   ; nur unteres Bytes
 move.w d0,6(a5)                 ; ins FDC-Array
 move.w #srcmd,dmamode           ; Command-Reg. selektieren
 rts

;***************** Track-Register lesen *********************

rtrkreg:

 move.w #srtrk,dmamode           ; Spur-Reg. selektieren
```

```
 bsr read1772                        ; und lesen
 and.w #$ff,d0                       ; nur unteres Byte
 move.w d0,4(a5)                     ; ins FDC-Array
 move.w #srcmd,dmamode               ; Command-Reg. selektieren
 rts

;********************* Status-Reg. lesen **********************

rstareg:

 move.w #srcmd,dmamode               ; Status-Reg. selektieren
 bsr read1772                        ; und lesen
 and.w #$ff,d0                       ; Status im unteren Byte
 move.w d0,12(a5)                    ; ins FDC-Array
 rts

;******************** Spur-Reg. schreiben *********************

wtrkreg:

 move.w #srtrk,dmamode               ; Spur-Reg. selektieren
 move.w 4(a5),d7                     ; Spur-Nr. holen
 and.w #$ff,d7
 bsr wrt1772                         ; und schreiben
 move.w #srcmd,dmamode               ; Command-Reg. selektieren
 rts

;******************** Set DMA-Transfer Adresse ****************

setdma:

 move.l d7,20(a5)                    ; Start-Adresse in FDC-Array retten
 move.b d7,dmalow                    ; erst das Low-Byte
 lsr.l #8,d7
 move.b d7,dmamid                    ; dann das Mid-Byte
 lsr.l #8,d7
 move.b d7,dmahigh                   ; und zuletzt das High-Byte schreiben

 move.l 20(a5),d7                    ; Start-Adresse zurückholen
 clr.l d6
```

```
 move.w 8(a5),d6                     ; Anzahl der zu Übertr. Bytes
 add.l d6,d7                         ; beides Addieren
 move.l d7,24(a5)                    ; = erwartete Endadresse
 rts

;*** DMA-Status lesen; Anzahl der übertragenen Bytes errechnen ***

readstat:

 move.w dmamode,d0                   ; DMA-Status lesen
 and.w #$7,d0                        ; nur die unteren 3 Bit nehmen
 move.w d0,14(a5)                    ; und nach fdcout

 clr.l d1                            ; DMA-Endadresse lesen
 move.b dmahigh,d1
 lsl.l #8,d1
 move.b dmamid,d1
 lsl.l #8,d1
 move.b dmalow,d1

 move.l d1,24(a5)                    ; End-Adresse ins Array
 sub.l 20(a5),d1                     ; End-Adr. minus Start-Adr.
 move.w d1,18(a5)                    ; =Anzahl der Bytes
 rts

;***************** FDC-Register schreiben ******************

wrt1772:

 bsr wait
 move.w d7,dmascnt                   ; FDC-Reg. bzw. DMA-Sectorcount
 bsr wait
 rts

;***************** FDC-Register lesen ******************

read1772:

 bsr wait
 move.w dmascnt,d0                   ; FDC-Reg. bzw. DMA-Sectorcount lesen
```

```
 bsr wait
 rts

;**************** Warten, bis FDC fertig ******************

fdcwait:

 move.l #$180,d5                ; etwas warten, bis Busy gesetzt
litlwt:
 dbra d5,litlwt

 move.l #$40000,d5              ; d5 als Timeout-Zähler
 cmp.w #$9,0(a5)                ; READ-ADDRESS-Kommando?
 bne readmfp
 move.l #$28000,d5              ; ja, kürzerer Timeout

readmfp:

 btst #5,mfp                    ; ist das Kommando beendet?
 beq fdcready                   ; ja

 subq.l #1,d5                   ; nein, Timeout-Zähler dekrementieren
 beq timeout                    ; falls abgelaufen

 tst.w dma-st(a3)               ; Kommando mit Datenübertragung?
 beq readmfp                    ; nein, weiter testen

 move.b dmahigh,temp+1-st(a3)   ; ist die erwartete DMA-Endadresse
 move.b dmamid,temp+2-st(a3)    ; schon erreicht?
 move.b dmalow,temp+3-st(a3)
 move.l temp-st(a3),d7
 cmp.l 24(a5),d7
 blt readmfp                    ; nein, weiter testen

 bsr force                      ; wenn ja, dann Kommando abbrechen
 move.w #0,dma-st(a3)           ; dma-Flag löschen
 bra fdcready                   ; und Routine normal beenden

timeout:
```

```
 move.w dmascnt,d0                  ; Status vor dem Abruch lesen
 and.w #$ff,d0                      ; oberes Byte ausblenden
 move.w d0,12(a5)                   ; und ins Array
 bsr force                          ; Kommando abbrechen
 move.w #1,16(a5)                   ; Timeoutflag setzen
 rts

fdcready:

 move.w dmascnt,d0                  ; Status lesen
 and.w #$ff,d0                      ; oberes Byte ausblenden
 move.w d0,12(a5)                   ; und ins FDC-Array
 rts

;************ warten, bis Motor ausgeschaltet *****************

motoroff:

 move.w #srcmd,dmamode              ; Statusreg. selektieren
test:
 bsr read1772                       ; und lesen
 btst #7,d0                         ; Motor-on gesetzt?
 bne test                           ; ja, weiter warten
 rts

;************************** Wait ******************************

wait:

 move.w sr,-(a7)                    ; Status retten
 move.w #$20,d5                     ; d5 als Zähler
wt2:
 dbf d5,wt2
 move.w (a7)+,sr                    ; Status zurückholen
 rts

;**************** Laufwerk & Seite selektieren ***************

setdrive:
```

```
 clr.l d7
 move.w 2(a5),d7                  ; Drive-Nr.holen
 bne set
 bsr motoroff                     ; falls 0, erst desel. wenn Motor aus
 move.w #1,vblflag-st(a3)         ; VBL-Rücksetzflag setzen
set:
 eor.b #7,d7                      ; Bits für Hardware invertieren
 and.b #7,d7                      ; nur die 3 Low-Bits beeinflussen
 move.w sr,-(a7)                  ; Status retten
 or.w #$700,sr                    ; Interrupts ausschalten
 move.b #$e,giselect              ; Port A des Sound-Chips selektieren
 move.b giselect,d0               ; Port A lesen
 and.b #$f8,d0                    ; Bits 0-2 löschen
 or.b d0,d7                       ; neue Bits setzen
 move.b d7,giwrite                ; und auf Port A schreiben
 move.w (a7)+,sr                  ; restore Status
 rts

;*******************************************************************
;******************** Variablen und Tabellen ********************
;*******************************************************************

 even

savreg: blk.l 16,0
savprm: dc.l 0
savstack: dc.l 0

vblflag: dc.w 0
dma: dc.w 0
temp: dc.l 0

functab: dc.l restore-st,seek-st
 dc.l step-st,stepin-st
 dc.l stepout-st,readsector-st
 dc.l writesector-st,readtrack-st
 dc.l writetrack-st,readaddress-st
 dc.l force-st,setdrive-st
 dc.l rsecreg-st,rtrkreg-st
 dc.l rstareg-st,wtrkreg-st
```

```
even

;***************************** ENDE ******************************
```

Doch kommen wir nun zum Listing des BASIC-Programms 'FDCCREAT.BAS'. Durch dieses Programm wird das File 'FDCINTER.IMG' erzeugt, welches später in ein BASIC-Programm eingebunden werden kann. Das funktioniert natürlich nicht nur mit dem ST-BASIC, sondern auch (oder gerade deshalb?!) mit einem 'vernünftigen' BASIC (z.B. GfA-BASIC).

```
************************************************************
****                                                    ****
****             LISTING => FDCCREAT.BAS                ****
****                                                    ****
************************************************************

10 '*************** FDCCREAT.BAS A.S. ***************
15 '
20 ?:fullw 2:clearw 2:gotoxy 0,0
25 ? "File >> fdcinter.img << wird erzeugt":?:?:?
30 dim c%( 688):cs#=0
35 for i=0 to 688
40 read a$:c%(i)=val("&H"+a$)
45 check#=check#+(c%(i))
50 next i
55 if check#= 2458472.96 then 70
60 ?"Geht leider noch nicht, da etwas mit den DATAs nicht stimmt."
65 goto 80
70 bsave "fdcinter.img",varptr(c%(0)), 1378
75 ? "Das Programm >> fdcinter.img << ist nun geschrieben."
80 ?:?:?:?"Bitte Taste drücken":a=inp(2):end
85 '
90 '********* DATAs für fdcinter.img *********
95 '
100 DATA 6042,0001,0011,0031,0051,0071,0090,00B0
```

```
101 DATA 00C0,00E0,00F0,00D0,0000,0000,0000,0000
102 DATA 0000,0000,0000,0000,0000,0000,0000,0000
103 DATA 0000,0000,0000,0000,0000,0000,0000,0000
104 DATA 0000,0000,4A6F,0004,6600,0074,47FA,FFB2
105 DATA 48EB,7FFF,04D2,42A7,3F3C,0020,4E41,5C8F
106 DATA 2740,0516,4BEB,0018,33FC,0001,0000,043E
107 DATA 3B7C,0000,0010,377C,0000,051C,377C,0000
108 DATA 051A,302D,0000,0280,0000,000F,E588,49EB
109 DATA 0522,2034,0800,4EB3,0800,4A6B,051A,6700
110 DATA 000A,33FC,0000,0000,043E,202B,0516,2F00
111 DATA 3F3C,0020,4E41,5C8F,4CEB,7FFF,04D2,4E75
112 DATA 33FC,0080,00FF,8606,3E2B,0002,6100,02EA
113 DATA 6100,0306,4E75,33FC,0086,00FF,8606,3E2D
114 DATA 0004,6100,02D4,33FC,0080,00FF,8606,3E2B
115 DATA 0004,6100,02C4,6100,02E0,4E75,33FC,0080
116 DATA 00FF,8606,3E2B,0006,6100,02AE,6100,02CA
117 DATA 4E75,33FC,0080,00FF,8606,3E2B,0008,6100
118 DATA 0298,6100,02B4,4E75,33FC,0080,00FF,8606
119 DATA 3E2B,000A,6100,0282,6100,029E,4E75,3E2B
120 DATA 0016,6100,0274,3E3C,0100,51CF,FFFE,4E75
121 DATA 2E2D,0020,6100,0202,377C,0001,051C,33FC
122 DATA 0090,00FF,8606,33FC,0190,00FF,8606,33FC
123 DATA 0090,00FF,8606,3E3C,000C,6100,023C,33FC
124 DATA 0084,00FF,8606,3E2D,0006,6100,022C,33FC
125 DATA 0080,00FF,8606,3E2B,000C,6100,021C,6100
126 DATA 0238,6100,01E0,4E75,286D,0028,2E2D,0024
127 DATA 6100,01A6,33FC,0090,00FF,8606,33FC,0190
128 DATA 00FF,8606,33FC,0090,00FF,8606,3E3C,0001
129 DATA 6100,01E6,33FC,0080,00FF,8606,382D,000A
130 DATA 0244,007F,3E2B,0010,6100,01CE,6100,01EA
131 DATA 18C0,4A6D,0010,56CC,FFEC,6100,0188,4E75
132 DATA 2E2D,001C,6100,0152,377C,0001,051C,33FC
133 DATA 0090,00FF,8606,33FC,0190,00FF,8606,33FC
134 DATA 0090,00FF,8606,3E3C,000E,6100,018C,33FC
135 DATA 0080,00FF,8606,3E2B,0012,6100,017C,6100
136 DATA 0198,6100,0140,4E75,2E2D,0020,6100,010A
137 DATA 377C,0001,051C,33FC,0190,00FF,8606,33FC
138 DATA 0090,00FF,8606,33FC,0190,00FF,8606,3E3C
139 DATA 000C,6100,0144,33FC,0184,00FF,8606,3E2D
140 DATA 0006,6100,0134,33FC,0180,00FF,8606,3E2B
```

```
141 DATA 000E,6100,0124,6100,0140,6100,00E8,4E75
142 DATA 2E2D,001C,6100,00B2,377C,0001,051C,33FC
143 DATA 0190,00FF,8606,33FC,0090,00FF,8606,33FC
144 DATA 0190,00FF,8606,3E3C,000E,6100,00EC,33FC
145 DATA 0180,00FF,8606,3E2B,0014,6100,00DC,6100
146 DATA 00F8,6100,00A0,4E75,33FC,0084,00FF,8606
147 DATA 6100,00D6,0240,00FF,3B40,0006,33FC,0080
148 DATA 00FF,8606,4E75,33FC,0082,00FF,8606,6100
149 DATA 00B8,0240,00FF,3B40,0004,33FC,0080,00FF
150 DATA 8606,4E75,33FC,0080,00FF,8606,6100,009A
151 DATA 0240,00FF,3B40,000C,4E75,33FC,0082,00FF
152 DATA 8606,3E2D,0004,0247,00FF,6100,006C,33FC
153 DATA 0080,00FF,8606,4E75,2B47,0014,13C7,FFFF
154 DATA 860D,E08F,13C7,FFFF,860B,E08F,13C7,FFFF
155 DATA 8609,2E2D,0014,4286,3C2D,0008,DE86,2B47
156 DATA 0018,4E75,3039,00FF,8606,0240,0007,3B40
157 DATA 000E,4281,1239,FFFF,8609,E189,1239,FFFF
158 DATA 860B,E189,1239,FFFF,860D,2B41,0018,92AD
159 DATA 0014,3B41,0012,4E75,6100,00CA,33C7,00FF
160 DATA 8604,6100,00C0,4E75,6100,00BA,3039,00FF
161 DATA 8604,6100,00B0,4E75,2A3C,0000,0180,51CD
162 DATA FFFE,2A3C,0004,0000,0C6D,0009,0000,6600
163 DATA 0008,2A3C,0002,8000,0839,0005,00FF,FA01
164 DATA 6700,005C,5385,6700,003C,4A6B,051C,6700
165 DATA FFE8,1779,FFFF,8609,051F,1779,FFFF,860B
166 DATA 0520,1779,FFFF,860D,0521,2E2B,051E,BEAD
167 DATA 0018,6D00,FFC4,6100,FD06,377C,0000,051C
168 DATA 6000,001C,3039,00FF,8604,0240,00FF,3B40
169 DATA 000C,6100,FCEA,3B7C,0001,0010,4E75,3039
170 DATA 00FF,8604,0240,00FF,3B40,000C,4E75,33FC
171 DATA 0080,00FF,8606,6100,FF50,0800,0007,6600
172 DATA FFF6,4E75,40E7,3A3C,0020,51CD,FFFE,46DF
173 DATA 4E75,4287,3E2D,0002,6600,000C,6100,FFD0
174 DATA 377C,0001,051A,0A07,0007,0207,0007,40E7
175 DATA 007C,0700,13FC,000E,00FF,8800,1039,00FF
176 DATA 8800,0200,00F8,8E00,13C7,00FF,8802,46DF
177 DATA 4E75,0000,0000,0000,0000,0000,0000,0000
178 DATA 0000,0000,0000,0000,0000,0000,0000,0000
179 DATA 0000,0000,0000,0000,0000,0000,0000,0000
180 DATA 0000,0000,0000,0000,0000,0000,0000,0000
```

```
181 DATA 0000,0000,0000,0000,0000,0000,0000,0000
182 DATA 0000,0000,00C0,0000,00D6,0000,00FC,0000
183 DATA 0112,0000,0128,0000,0150,0000,0248,0000
184 DATA 0200,0000,02A0,0000,01A8,0000,013E,0000
185 DATA 0492,0000,02E8,0000,0306,0000,0324,0000
186 DATA 033A
```

Falls ihnen bei der Eingabe des Listings keine Fehler unterlaufen sind, verfügen nun über das Maschinen-Programm 'FDCINTER.IMG', welches es ermöglicht, alle FDC-Kommandos (und noch einiges mehr) aufzurufen.

Wie die Einbindung in ein BASIC-Programm stattfindet, ist den Kommentaren im ersten Teil des im nächsten Kapitel folgenden Listings zu entnehmen. Das ist so einfach und unproblematisch, daß es hier keiner weiteren Erklärung dazu bedarf. Man kann diesen Teil sogar noch verkürzen. Wenn z.B. nicht alle Informationen einer Spur gleichzeitig zur Verfügung stehen müssen, reicht es aus, an den Stellen, wo die Start-Adressen der einzelnen Puffer übergeben werden, jeweils die Start-Adresse des selben Puffers einzutragen.

**Die Übergabe der Parameter an 'FDCINTER.IMG'**

Bevor nun Demo-Programme folgen, möchten wir Sie umfassend mit dem Maschinen-Programm vertraut machen. Wir gehen davon aus, daß Sie dieses Programm, vielleicht auch nur einzelner Funktionen wegen, in ihre BASIC-Programme integrieren möchten. Für ein solches Vorhaben ist es zu mühselig, die dazu nötigen Informationen in irgendwelchen Listings zu suchen. Eine gute Beschreibung des Maschinen-Programms macht eine solche Suche überflüssig.

Beginnen wir mit einem Überblick, der darlegt, für welche Kommandos welche Parameter übergeben werden und fügen diesem einige Erklärungen an.

```
*************************************************************
***  Tabelle: "Eingabe-Parameter; Disk-File:EINGABE.GEM  ***
*************************************************************
```

(1) In FDC%(15) wird die Nummer des ersten zu lesenden bzw. des ersten zu schreibenden Sektors eingetragen. Beachten Sie dabei bitte, daß sich diese Angabe immer auf die Spur bezieht, über der sich der Schreib/Lese-Kopf momentan befindet. Falls es ihr Wunsch ist, mit 'logischen' Sektor-Nummern zu arbeiten, so müssen diese zuerst in absolute Spur/Sektor-Adressen umgerechnet werden.

(2) Die Anzahl der zu lesenden oder zu schreibenden Sektoren wird indirekt, über die Angabe der zu übertragenden Bytes, in FDC%(16) eingetragen. Das erscheint zunächst etwas umständlich, hat aber den Vorteil, daß auch Formate mit unterschiedlichen Sektorgrößen korrekt gelesen oder geschrieben werden können.

Sollen z.B. 5 Sektoren des ATARI-Formats übertragen werden, so trägt man einen Wert von 5*512 in FDC%(16) ein. Bei einem Format mit einer Sektorgröße von 256 Bytes würde bei gleicher Anzahl der Sektoren, 5*256 übergeben. Es ist auch möglich diesen 'multiplen Sektorzugriff' bei Formaten, in denen unterschiedliche Sektorgrößen in einer Spur vorkommen, anzuwenden. Wenn z.B bei einem 'Kopierschutz-Format', 4 aufeinanderfolgende Sektoren gelesen werden sollen, deren Sektorgrößen 1024, 512, 256 und 128 Bytes betragen (egal in welcher Reihenfolge) so reicht die Übergabe von '1024+512+256+128' in FDC%(16). Es ist auch möglich, z.B. nur die Hälfte eines Sektors zu lesen oder zu schreiben. Die entsprechende Anzahl zu übertragender Bytes braucht nur eingetragen werden.

Das gleiche betrifft die Kommandos READ-TRACK und WRITE- TRACK. Soll durch den Aufruf eine komplette Spur bearbeitet werden, so übergeben Sie einen Wert >6300. Mit einem geringeren Wert kann man erreichen das z.B. nur ein Teil einer Spur formatiert wird. Durch mehrfaches Formatieren lassen sich Besonderheiten in der Spur erzeugen, die in den Kreisen der 'Software-Cracker' den Kommentar:"wieder ein neuer, mieser Trick" hervorrufen.

>>> Sehr wichtig <<< : Werden Daten geschrieben (WRITE- SECTOR, WRITE-TRACK) muß zu der Anzahl zu übertragender Bytes, unbedingt 32 ($20) addiert werden. Der DMA- Controller holt sich nämlich vorsorglich, um auf die Datenübertragung vorbereitet zu sein, 32 Byte in seine internen Register. So sind z.B. die Daten für zwei Sektoren (2*512 Byte= $400) erst übertragen, wenn die DMA-End-Adresse gegenüber der DMA-Start-Adresse um $420 erhöht ist.

(3) Die Anzahl der zu lesenden ID-Felder ist auf 128 begrenzt. Das reicht im allgemeinen aus, um auch bei den 'merkwürdigsten' Formaten alle Id-Felder lesen zu können. In FDC%(17) wird immer die Anzahl-1 eingetragen. Es müssen mindestens 3 Felder gelesen werden, bevor durch den DMA-Controller Daten in den Puffer transferiert werden. Damit die gesamte ID-Information in den Speicher gebracht wird, muß der Wert - #ID-Felder*6 - durch 16 teilbar sein.

(siehe auch: 'FDC-Kommandobeschreibung: READ-ADDRESS')

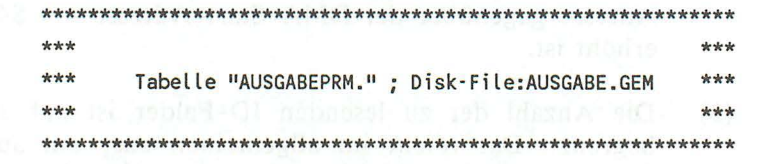
(4) Die Start-Adressen der Puffer müssen nicht bei jedem Aufruf angeben werden. Im allgemeinen geschieht das - wie in unserem Demo-Programm - nur einmal. Sollen die Informationen von mehreren Spuren gleichzeitig im Speicher gehalten werden, so di-

mensionieren Sie einfach weitere Arrays, deren Start-Adresse vor Aufruf übergeben wird. Da bei den Adressen immer 'Langworte' übergeben werden, erreicht man dies sehr einfach durch 'POKE'. Nehmen wir z.B. einen zweiten Sektor-Puffer, dann sieht das folgendermaßen aus:

```
dim sec2%(3200):def seg=0:poke fdc#+56,varptr(sec2
%(0))
```

**Die Ausgabe der Parameter von 'FDCINTER.IMG'**

Natürlich erhalten wir von dem Maschinenprogramm zahlreiche Parameter zurück. Diese haben wir ebenfalls in einer Tabelle aufgeführt.



```
*********************************************************
***                                                   ***
***   Tabelle "AUSGABEPRM." ; Disk-File:AUSGABE.GEM   ***
***                                                   ***
*********************************************************
```

In dieser Übersicht finden Sie die beiden Array-Elemente FDC%(14) und FDC%(15), die schon bei der Übergabe Verwendung fanden, wieder. Diese bilden bei der Parameter-Behandlung die einzige Ausnahme. Ansonsten sind Ein- und Ausgabe-Prm. streng voneinander getrennt. D.h.: Die Übergabe-Parameter werden, außer bei 'Sektor-Register lesen' und 'Spur-Register lesen', in keiner Weise von dem Maschinenprogramm geändert.

Zum FDC-Status [FDC%(18)] empfehlen wir, die erforderlichen Informationen dem Kapitel 'Der FDC-Status nach Kommandos' zu entnehmen. Natürlich hilft hier auch die allgemeine Beschreibung der FDC-Kommandos weiter.

Der DMA-Status [FDC%(19)] ist leicht zu erklären. Hier sind nur 3 Bits interessant. Bit 0 ist gesetzt wenn kein Fehler bei dem

DMA-Transfer auftrat. Bit 1 ist gesetzt wenn der Inhalt des 'Sector-Count-Registers' im DMA-Controller nicht auf 0 heruntergezählt wurde. Dem DMA-Controller wird über dieses Register mitgeteilt, welche maximale Anzahl von Daten ab der Start-Adresse gerechnet übertragen werden dürfen. Um diese Angabe brauchen Sie sich jedoch nicht zu kümmern, da diese Aufgabe der Maschinen-Routine zufällt. Bit 2 ist eine 'Kopie' des DRQ-Ausgangs des FDC. Nach einem Kommando mit Daten-Transfer sind, bei fehlerfreier Ausführung, Bit 0 und Bit 1 gesetzt. Sie finden in FDC%(19) also eine '3'. Falls das einmal nicht der Fall sein sollte, werden Sie im FDC-Status gleichzeitig auch das LOST-DATA-Bit gesetzt finden.

Die DMA-Start-Adresse enthält jeweils die Adresse des aktuellen Puffers. Nach einem READ-SECTOR-Kommando würde hier demnach die Start-Adresse des Sektor-Puffers zu finden sein.

Die DMA-End-Adresse spiegelt den Puffer-Zeiger des DMA-Controllers wider. Diese Adresse minus der Start-Adresse wird als Anzahl der übertragen Bytes in FDC%(21) ausgegeben. Der Interpretation dieser Angaben ist erhöhte Aufmerksamkeit zu schenken. In Lese-Richtung wird der Zeiger nach Erhalt von 16 ($10) Daten-Bytes um diesen Wert erhöht und die Daten-Bytes, die bis zu diesem Zeitpunkt im DMA-Controller zwischengespeichert wurden, in den Puffer transferiert. In Schreib-Richtung werden vor dem Daten-Tranfer 32 ($20) Bytes in diese internen Zwischenspeicher geholt und der Puffer-Zeiger um diesen Wert erhöht.

Ein Timeout (FDC%(20)=1) dürfte eigentlich selten auftreten, da die Wartezeit der Maschinen-Routine großzügig bemessen wurde und der FDC in dieser Zeit schon selbsttätig ein Kommando - im Fehlerfall - abgebrochen hat. Da der FDC dies erst nach ca. 1,5 s veranlaßt, wurde die Wartezeit für das READ-ADDRESS-Kommando jedoch kürzer ausgelegt. Der Grund dafür ist folgender: Sie möchten 100 ID-Felder lesen (ob das sinnvoll ist lassen wir dahingestellt) und übergeben deshalb vor Aufruf des READ-ADDRESS-Kommandos in FDC%(17) den Wert '99'. Das Maschinen-Programm führt das READ-ADDRESS-Kommando

demnach 100 mal aus, bevor es zum BASIC zurückkehrt. Falls der FDC keine ID-Felder finden sollte (und das 100 mal) müßten Sie über 2 min. auf diese Rückkehr warten. So etwas verleitet zum Ausschalten des Rechners und das möchten wir nicht provozieren. Sollte der FDC nicht in angemessener Zeit ein ID-Feld lesen können wird er mittels FORCE-INTERRUPT unterbrochen und das Maschinenprogramm kehrt zum BASIC zurück.

**Die Kommandoworte für den FDC**

Ein weiterer Punkt, der unserer FDC-Interface erst richtig universell macht, wurde bis jetzt nur am Rande erwähnt. Es geht um die Kommandoworte, die dem FDC übergeben werden. Es wäre schade, könnte man diese nicht den eigenen Wünschen entsprechend anpassen. Doch auch diese Möglichkeit haben wir vorgesehen. Wo sich die Kommando-Worte befinden und mit welchen Werten sie initialisiert sind zeigt die Übersicht.

| Kommandowort für | befindet sich in | ist initialisiert mit |
|---|---|---|
| RESTORE | FDC%(1) | $01 |
| SEEK | FDC%(2) | $11 |
| STEP | FDC%(3) | $31 |
| STEP-IN | FDC%(4) | $51 |
| STEP-OUT | FDC%(5) | $71 |
| READ-SECTOR | FDC%(6) | $90 |
| WRITE-SECTOR | FDC%(7) | $B0 |
| READ-ADDRESS | FDC%(8) | $C0 |
| READ-TRACK | FDC%(9) | $E0 |
| WRITE-TRACK | FDC%(10) | $F0 |
| FORCE-INTERRUPT | FDC%(11) | $D0 |

Die genaue Bedeutung der 'Option-Bits' in den Kommando-Worten, kann in der Kommandobeschreibung des FDC nachge-

lesen werden. Beachten Sie, das bei READ-SECTOR und WRITE-SECTOR das m-Bit (für multi-Sektor-Read/Write) gesetzt ist. Wenn Sie hier Änderungen vornehmen und dieses Bit dabei löschen, wird nur noch jeweils ein Sektor bearbeitet.

### 8.3.2 Demo 1 - Alle FDC-Kommandos im Griff

Nach soviel Information zu einer vergleichsweise kleinen Maschinen-Routine ist es endlich an der Zeit, zu sehen ob sie auch tatsächlich unseren Ansprüchen gerecht wird.

Wie schon im vorherigen Kapitel erwähnt, besteht dieses Listing aus zwei Teilen, wobei der erste Teil nur zeigen soll, wie einfach die Einbindung in eigene Programme erfolgt. Wenden wir uns also dem zweiten Teil zu.

Dieser Teil des Listings ist ein Programm, welches die Aufgabe hat, in der Praxis zu zeigen, wie die Parameter, vor dem Aufruf einer Funktion, an die Maschinen-Routine übergeben werden. Ferner hat es einen echten 'Demo-Charakter', da es den direkten Zugriff auf alle Kommandos des FDC erlaubt und im Anschluß an einem solchen die vollständige Information - vom Status bis zu den Daten - anzeigt. Sie können also nach Herzenslust mit dem Floppy-Controller experimentieren. Wenn Fragen irgendwelcher Art auftauchen - das Kapitel über den FDC enthält mit Sicherheit die Antwort darauf.

Das Programm besteht in der Hauptsache aus einem 'Info-Bildschirm' der in zwei Teile gegliedert ist:

1. In dem oberen Teil sind 20 Funktionen aufgeführt, von denen die ersten sechszehn (0-15) jene sind, welche unsere FDC-Schnittstelle in der Lage ist auszuführen.

   Bevor ein FDC-Kommando (Funktionen 0-10) aufgerufen wird, muß zuerst ein Laufwerk selektiert werden. Dies geschieht über die Funktion 11. Der Übergabe-Wert dazu ist:

2 => für Laufwerk A, Seite 0
3 => für Laufwerk A, Seite 1
4 => für Laufwerk B, Seite 0
5 => für Laufwerk B, Seite 1
oder 0 => für deselektieren

Wenn Sie das Programm über Funktion 19 (Ende) beenden, wird das deselektieren der Laufwerke automatisch ausgeführt.

Da von einigen Funktionen Daten übertragen werden, wäre es schade, wenn man sich diese nicht anschauen könnte. Es wurden deshalb noch die Menüpunkte 16-18 zugefügt, wodurch ein Betrachten der Daten ermöglicht wird.

Nebenbei bemerkt; erweitern Sie dieses 'Demo-Programm' doch um die Funktion, die Puffer-Daten ändern zu können. Sie besitzen damit einen Disk-Monitor, der mit 'Features' aufwartet, die Sie woanders wahrscheinlich vergeblich suchen.

2. Der untere Teil des Bildschirms enthält sämtliche Parameter die an die Unter-Routine bzw. von der Unter-Routine übergeben werden. Obendrein werden noch die Start-Adressen aller Puffer angezeigt.

   Auf den ersten Blick ist diese 'geballte' Information etwas verwirrend und läßt die Aufrufe des Maschinen-Programms komplizierter erscheinen als sie tatsächlich sind. Wenn Sie die Tabellen der Ein- und Ausgabe-Parameter zu Rate ziehen und die, für einen Aufruf relevanten Parameter betrachten, werden Sie erkennen, daß neben der Funktions-Nr. nur höchstens zwei weitere Parameter - läßt man die Start-Adressen der Puffer einmal außer acht - übergeben werden müssen. Für die Hälfte der Funktio-

nen reicht sogar die Übergabe der Funktions-Nummer vollkommen aus. Einfacher geht es wohl kaum.

Als letzte Programm-Information dazu ist noch zu sagen, daß die Parameter, die für die einzelnen Funktionen benötigt werden, vom Programm abgefragt werden, wobei die vorherigen Werte angezeigt werden. Sollen diese nochmals Verwendung finden, so reicht ein Druck auf die 'Return'-Taste dazu aus.

```
*************************************************************
***                                                       ***
***                Listing: FDCINTER.BAS                  ***
***                                                       ***
*************************************************************
```

```
1000 '******************************************************************
1010 '*****      FDC-Interface für BASIC (Teil 1) A.S. (7/86)      *****
1020 '******************************************************************
1030 '
1040 ' Es gibt nur wenig vorzubereiten, um die FDC-Routinen installieren
1050 ' zu können. Auf jeden Fall benötigen wir etwas Speicher, in den
1060 ' zum einen die Routinen selbst und zum anderen die Daten, welche
1070 ' für die Disketten-Operationen nötig sind, abgelegt werden. Dazu
1080 ' dimensionieren wir ein paar Integer-Arrays und holen uns deren
1090 ' Start-Adressen.
1100 '
1110 dim fdc%(700) :fdc# =varptr(fdc%(0))
1120 dim trk%(3200):trk# =varptr(trk%(0))
1130 dim sec%(2600):sec# =varptr(sec%(0))
1140 dim adr%(768) :adr# =varptr(adr%(0))
1150 dim stat%(64) :stat#=varptr(stat%(0))
1160 '
1170 ' In das fdc%-Array werden die Floppy-Routinen geladen .....
1180 '
1190 bload "fdcinter.img",fdc#
1200 '
1210 ' und die Start-Adressen der anderen Arrays eingePOKEt.
1220 '
```

```
1230 def seg = 0 :' Wir POKEn Langworte
1240 '
1250 poke fdc#+52,trk# :' Spur-Puffer
1260 poke fdc#+56,sec# :' Sektor-Puffer
1270 poke fdc#+60,adr# :' ID-Feld-Puffer
1280 poke fdc#+64,stat#:' ID-Status-Puffer
1290 '
1300 ' Das war schon alles! Wie man nun die einzelnen Funktionen auf-
1310 ' ruft, wird im folgenden Teil erklärt.
1320 '
1330 '*****************************************************************
1340 '*********************     TEIL 2     ******************************
1350 '*****************************************************************
1360 '
1370 'Dies ist nur ein kleines Demo, welches wir aber dennoch nicht all-
1380 'zu spartanisch gestalten möchten. Beginnen wir mit einem kleinen
1390 'Menü, das die einzelnen Funktions-Nr. und alle Parameter anzeigt.
1400 'Die Funktionen 0-15 sind die, die von der Maschinen-Routine bear-
1410 'beitet werden, während 16-18 nur zum Betrachten der Puffer dienen.
1420 'Diese Funktionen können Sie natürlich weitaus komfortabler gestal-
1430 'ten. Funktion 19 beendet dieses Demo und deselektiert die Lauf-
1440 'werke.
1450 '
1460 '
1470 fullw 2 : width 255
1480 Menu:
1490 ? : clearw 2 : gotoxy 0,0
1500 '
1510 ?" ------------------------ Verfügbare Funktionen -----------";
1520 ?"-------------"
1530 ?" 0 => Restore              1 => Seek               2 => Step "
1540 ?" 3 => Step-in              4 => Step-out           5 => Read-S";
1550 ?"ector"
1560 ?" 6 => Write-Sector         7 => Read-Track         8 => Write-";
1570 ?"Track"
1580 ?" 9 => Read-Address         10=> Force-Interrupt    11=> Select";
1590 ?" Drive"
1600 ?" 12=> Sektorreg. lesen     13=> Spurreg. lesen     14=> Status";
1610 ?"reg. lesen"
1620 ?" 15=> Spur-Reg. schreiben 16=> Spur-Puffer zeigen  17=> Sektor";
```

```
1630 ?"-Puffer zeigen"
1640 ?" 18=> ID-Felder zeigen     19=> Programm beenden":?
1650 '
1660 ?" ------------------------ Anzeige aller Parameter -----------";
1670 ?"------------"
1680 '
1690 ?" Funktion :              FDC-Status :$          Spur-Puffer     :$"
1700 ?" Laufwerk :              DMA-Status :$          Sektor-Puffer   :$"
1710 ?" Spur     :              Timeout    :$          ID-Feld-Puffer  :$"
1720 ?" Sektor   :              DMA-Start  :$          ID-Feld-Status  :$"
1730 ?" #Bytes   :$             DMA-Ende   :$"
1740 ?" #Id's -1 :              #DMA-Bytes :$"
1750 ?" -----------------------------------------------------------------";
1760 ?"------------"
1770 '
1780 main:
1790
'==========================================================================
1800 gotoxy 06,10 :?right$("   "+str$(fdc%(12)),4)
1810 gotoxy 06,11 :?right$("   "+str$(fdc%(13)),4)
1820 gotoxy 06,12 :?right$("   "+str$(fdc%(14)),4)
1830 gotoxy 06,13 :?right$("   "+str$(fdc%(15)),4)
1840 gotoxy 06,14 :?right$("    "+hex$(fdc%(16)),4)
1850 gotoxy 06,15 :?right$("    "+str$(fdc%(17)),4)
1860 gotoxy 17,10 :?right$("      "+hex$(fdc%(18)),6)
1870 gotoxy 17,11 :?right$("      "+hex$(fdc%(19)),6)
1880 gotoxy 17,12 :?right$("      "+hex$(fdc%(20)),6)
1890 gotoxy 17,13 :?right$("      "+hex$(fdc%(22))+hex$(fdc%(23)),6)
1900 gotoxy 17,14 :?right$("      "+hex$(fdc%(24))+hex$(fdc%(25)),6)
1910 gotoxy 17,15 :?right$("      "+hex$(fdc%(21)),6)
1920 gotoxy 30,10 :?right$("      "+hex$(fdc%(26))+hex$(fdc%(27)),6)
1930 gotoxy 30,11 :?right$("      "+hex$(fdc%(28))+hex$(fdc%(29)),6)
1940 gotoxy 30,12 :?right$("      "+hex$(fdc%(30))+hex$(fdc%(31)),6)
1950 gotoxy 30,13 :?right$("      "+hex$(fdc%(32))+hex$(fdc%(33)),6)
1960 '
1970 gotoxy 0,17:?spc(220);
1980 key:
1990 gotoxy 1,17:?spc(50)
2000 gotoxy 1,17:input " Welche Funktion";func$:func=val(func$)
2010 if func<0 or func>19 then menu
```

```
2020 func=func+1
2030 if func=20 then fdc%(12)=11:fdc%(13)=0:call fdc#:end
2040 '
2050 if func<17 then 2110
2060 reset
2070 func=func-16:clearw 2:
2080 on func gosub dumptrk,dumpsec,dumpid
2090 openw 2:goto key
2100 '
2110 on func gosub a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p
2120 gotoxy 1,19:?"Funktion ausführen (j/n) ?";
2130 if chr$(inp(2))<>"j" then main
2140 call fdc#
2150 goto main
2160 '
2170 '*********************************************************************
2180 '*********************************************************************
2190 '
2200 'Es folgen die 16 Funktionen, welche von unserer Maschinen-Routine
2210 'unterstützt werden. Hier sehen Sie, welche Parameter vor dem Auf-
2220 'ruf "call fdc#" gesetzt werden müssen. In vielen Fällen reicht es
2230 'aus, die Funktionsnr. in fdc%(12) zu übergeben.
2240 '
2250 '
2260 '==================== RESTORE =================================
2270 a:
2280 fdc%(12)=0
2290 gotoxy 1,17:?"RESTORE - keine Parameter nötig";:return:
2300 '
2310 '==================== SEEK ===================================
2320 b:
2330 fdc%(12)=1
2340 gotoxy 1,17:?"SEEK - Welche Spur-Nr.(alt=>";
2350 ?fdc%(14);")";:input v$:if len(v$)=0 then return
2360 fdc%(14)=val(v$):return
2370 '
2380 '==================== STEP ===================================
2390 c:
2400 fdc%(12)=2
2410 gotoxy 1,17:?"STEP - keine Parameter nötig";:return
```

```
2420 '
2430 '==================== STEP_IN =================================
2440 d:
2450 fdc%(12)=3
2460 gotoxy 1,17:?"STEP IN - keine Parameter nötig";:return
2470 '
2480 '==================== STEP_OUT ================================
2490 e:
2500 fdc%(12)=4
2510 gotoxy 1,17:?"STEP OUT - keine Parameter nötig";:return
2520 '
2530 '==================== READ_SECTOR(s) ===========================
2540 f:
2550 fdc%(12)=5
2560 gotoxy 1,17:?"READ SECTOR - Welcher Startsektor (alt=>";
2570 ?fdc%(15);")";:input v$:if len(v$)=0 then 2590
2580 fdc%(15)=val(v$)
2590 gotoxy 1,18:?"Anzahl der Bytes (alt=>$";hex$(fdc%(16));")";
2600 input v$:if len(v$)=0 then return
2610 fdc%(16)=val(v$):return
2620 '
2630 '==================== WRITE_SECTOR(s) ==========================
2640 g:
2650 fdc%(12)=6
2660 gotoxy 1,17:?"WRITE SECTOR - Welcher Startsektor (alt=";
2670 ?">";fdc%(15);")";:input v$:if len(v$)=0 then 2690
2680 fdc%(15)=val(v$)
2690 gotoxy 1,18:?"Anzahl der Bytes (alt=>$";hex$(fdc%(16));")";
2700 input v$:if len(v$)=0 then return
2710 fdc%(16)=val(v$):return
2720 '
2730 '==================== READ_TRACK ============================
2740 h:
2750 fdc%(12)=7
2760 gotoxy 1,17:?"READ TRACK - Anzahl der Bytes (alt=>$";
2770 ?hex$(fdc%(16));")";:input v$:if len(v$)=0 then return
2780 fdc%(16)=val(v$):return
2790 '
2800 '==================== WRITE_TRACK ===========================
2810 i:
```

```
2820 fdc%(12)=8
2830 gotoxy 1,17:?"WRITE TRACK - Anzahl der Bytes (alt=>$";
2840 ?hex$(fdc%(16));")";:input v$:if len(v$)=0 then return
2850 fdc%(16)=val(v$):return
2860 '
2870 '==================== READ_ADDRESS ============================
2880 j:
2890 fdc%(12)=9
2900 gotoxy 1,17:?"READ ADDRESS - Anzahl der ID-Felder-1 (alt=>";
2910 ?fdc%(17);")";:input v$:if len(v$)=0 then return
2920 fdc%(17)=val(v$):return
2930 '
2940 '=================== FORCE INTERRUPT ============================
2950 k:
2960 fdc%(12)=10
2970 gotoxy 1,17:?"FORCE INTERRUPT - keine Parameter nötig";:return
2980 '
2990 '================= Laufwerk selektieren ========================
3000 l:
3010 fdc%(12)=11:gotoxy 1,17
3020 ?"(X=Lfw/Seite) : 2=A/0; 3=A/1; 4=B/0; 5=B/1; 0=deselektieren"
3030 gotoxy 1,18:?"Welches Laufwerk (alt=>";fdc%(13);")";
3040 input v$:if len(v$)=0 then return
3050 fdc%(13)=val(v$):return
3060 '
3070 '================= Sektor-Register lesen ========================
3080 m:
3090 fdc%(12)=12
3100 gotoxy 1,17:?"SEKTOR-REGISTER LESEN - keine Parameter nötig";
3110 return
3120 '
3130 '================= Spur-Register lesen ========================
3140 n:
3150 fdc%(12)=13
3160 gotoxy 1,17:?"SPUR-REGISTER LESEN - keine Parameter nötig";
3170 return
3180 '
3190 '================= Status-Register lesen ========================
3200 o:
3210 fdc%(12)=14
```

```
3220 gotoxy 1,17:?"STATUS-REGISTER LESEN - keine Parameter nötig";
3230 return
3240 '
3250 '================== Spur-Register schreiben =====================
3260 p:
3270 fdc%(12)=15
3280 gotoxy 1,17:?"SPUR-REGISTER SCHREIBEN - Welche Spur-Nr.(alt=>";
3290 ?fdc%(14);")";:input v$:if len(v$)=0 then return
3300 fdc%(14)=val(v$):return
3310 '
3320 '*******************************************************************
3330 '*******************************************************************
3340 '
3350 'Die folgenden Funktionen (16-18) haben nichts mit der Maschinen-
3360 'Routine zu tun, sondern sollen nur die Puffer-Inhalte auflisten.
3370 '
3380 '================== Spur-Puffer anzeigen ========================
3390 dumptrk:
3400 gotoxy 0,0:?"SPUR-PUFFER ZEIGEN (alle Werte=>hex, (w)eiter, ";
3410 ?"(e)nde)":?
3420 ?" PUFFER 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 01234";
3430 ?"56789ABCDEF"
3440 ?" ------------------------------------------------------ -----";
3450 ?"-----------"
3460 ch$=" "
3470 '
3480 lcnt=0
3490 for id = 1 to (fdc%(16)+1)-16 step 16
3500 lcnt=lcnt+1
3510 id%(1)=id-1:?" +";right$("0000"+hex$(id%(1)),4);" ";
3520 for by=0 to 15:def seg =id+by:id%(1)=peek(trk#-1)
3530 ?right$("00"+hex$(id%(1)),2);" ";
3540 if id%(1)=7 or id%(1)=10 or id%(1)=13 then id%(1)=20
3550 mid$(ch$,by+1,1)=chr$(id%(1)):next by:?" ";ch$
3560 '
3570 if lcnt<10 then 3610
3580 lcnt=0:dum=inp(2)
3590 if chr$(dum)="e" then id=70000:goto 3610
3600 if chr$(dum)<>"w" then 3580
3610 next id
```

```
3620 ?"Fertig! Bitte Taste drücken...";
3630 dum=inp(2):return
3640 '
3650 '================= Sektor-Puffer anzeigen ========================
3660 dumpsec:
3670 lcnt=0
3680 gotoxy 0,0:?"SEKTOR-PUFFER ZEIGEN (alle Werte=>hex, (w)eiter,";
3690 ?" (e)nde)":?
3700 ?" PUFFER 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 01234";
3710 ?"56789ABCDEF"
3720 ?" -------------------------------------------------------------";
3730 ?"-----------"
3740 '
3750 ch$=" "
3760 for id = 1 to (fdc%(16)+1)-16 step 16
3770 lcnt=lcnt+1
3780 id%(1)=id-1:?" +";right$("0000"+hex$(id%(1)),4);" ";
3790 for by=0 to 15:def seg =id+by:id%(1)=peek(sec#-1)
3800 ?right$("00"+hex$(id%(1)),2);" ";
3810 if id%(1)=7 or id%(1)=10 or id%(1)=13 then id%(1)=20
3820 mid$(ch$,by+1,1)=chr$(id%(1)):next by:?" ";ch$
3830 '
3840 if lcnt<10 then 3880
3850 lcnt=0:dum=inp(2)
3860 if chr$(dum)="e" then id=70000:goto 3880
3870 if chr$(dum)<>"w" then 3850
3880 next id
3890 ?"Fertig! Bitte Taste drücken...";
3900 dum=inp(2):return
3910 '
3920 '=================== ID-Felder anzeigen ========================
3930 dumpid:
3940 gotoxy 0,0:?"ID-FELDER ZEIGEN (alle Werte=>hex, (w)eiter, (e)nde)"
3950 ?:?" PUFFER SPUR SEITE SEKTOR LÄNGE CRC1 CRC2 FDC-Status"
3960 ?" ----------------------------------------------------------"
3970 '
3980 lcnt=0
3990 for id = 1 to (fdc%(17)+1)*6 step 6
4000 lcnt=lcnt+1
4010 id%(1)=id-1:?" +";right$("00"+hex$(id%(1)),2);" ";
```

```
4020 for by=0 to 5:def seg =id+by:id%(1)=peek(adr#-1)
4030 ?right$("00"+hex$(id%(1)),2);" ";:next by
4040 def seg=id/6+1:id%(1)=peek(stat#-1)
4050 ?" ";right$("00"+hex$(id%(1)),2)
4060 '
4070 if lcnt<9 then 4110
4080 lcnt=0:dum=inp(2)
4090 if chr$(dum)="e" then id=1000:goto 4110
4100 if chr$(dum)<>"w" then 4080
4110 next id
4120 ?"Fertig! Bitte Taste drücken...";
4130 dum=inp(2):return
4140 '
4150 '======================== ENDE ==================================
```

### 8.3.3 Demo 2 - Disketten kopieren

Mit dem folgenden kleinen BASIC-Programm möchten wir Ihnen eine andere Anwendung unserer FDC-Maschinen-Routinen zeigen und damit demonstrieren, wie einfach der direkte Zugriff auf den Floppy-Controller, von BASIC aus geschehen kann. Das Ergebnis ist ein Kopierprogramm, mit dem Sie eine zweiseitige Diskette in ca. <<< 85 Sekunden >>> duplizieren können!

Der Aufwand, ein solches Programm zu schreiben, ist wirklich nicht der Rede wert. Die Einschränkung bei diesem Programm ist allerdings, daß es nur mit zwei Laufwerken funktioniert. Das liegt darin begründet, daß wir (der Umstände wegen) nicht zu viele Puffer benutzen wollten. Denn in ST-BASIC darf die Größe eines Arrays maximal 32-kB betragen.

Das verhindert natürlich ein 'dim sec%(79,2303)', womit man Platz für die Hälfte einer doppelseitigen Diskette hätte. Man könnte das gleiche natürlich mit 'dim sec1%(2303), sec2%(2303), ....' erreichen, wozu wir aber keine Lust verspürten. Es sollte ja schließlich ein 'Demo' entstehen und die Programmierfreude, Bestehendes noch erheblich zu verbessern, wollten wir ihnen nicht nehmen.

Aber sei's drum. Wir speichern jedenfalls nur die Information von zwei Spuren (jeweils Vor- und Rückseite) und schreiben diese auf die gleiche Spur der Ziel-Diskette. Dieser Ablauf würde, bei Verwendung eines Laufwerks, 80 Diskettenwechsel zur Folge haben. Das ist natürlich unzumutbar.

Das DESK-TOP benötigt für das Kopieren einer zweiseitigen Diskette ca.195 Sekunden. Diese Zeit ist damit um 50% höher, als unser Programm (130 Sek.) für die Erledigung der gleichen Aufgabe braucht.

Dieser Wert ist erstaunlich gering, wenn man berücksichtigt, daß ständig die aktuelle Spur angezeigt wird. In der FDC-Unterroutine besteht ferner ein 'Overhead', der bei jedem Aufruf durchlaufen wird (z.B. ein- und ausschalten des SUPERVISOR-MODE). Letztendlich verschlingt auch das ständige Umselektieren der beiden Laufwerke einige Zeit. Trotz dieser widrigen Umstände sind 130 sek. doch wirklich akzeptabel.

Ein echtes 'Tune Up' läßt sich erreichen, wenn die, im Vergleich zur Kopierzeit, extrem langsamen PRINT-Befehle (Zeile 60, 66, 76 und 82) aus der Kopierschleife entfernt werden. In diesem Fall bedeutet der Verzicht auf den Anzeige-Komfort, einen Zeitgewinn von 45 Sekunden! Anders ausgedrückt; die Kopierzeit beträgt nur noch 85 Sekunden!

In diesem Programm tritt ein Fall ein, der ein Ändern des Kommando-Wortes, zumindest aber ein Schreiben des Spur-Registers erforderlich macht.

Beginnen wir im Ablauf des Programms bei Spur-0. Im Anschluß an das Lesen dieser Spur (Vor- und Rückseite) wird für Laufwerk A ein STEP-IN ausgeführt. Der Schreib/Lese-Kopf befindet sich in diesem Laufwerk danach über Spur-1 und auch das Spur-Register des FDC enthält eine '1'.

Nun soll die zuvor gelesene Spur-0 auf Laufwerk B geschrieben werden. Wird an dieser Stelle nicht korrektiv eingegriffen, geschieht schlicht und ergreifend folgendes:

Der FDC wird das WRITE-SECTOR-Kommando mit 'Record-not-found-Error' abbrechen, da das Spur-Register auf '1' steht, die ID-Felder, als Wert für die Spur-Nr., jedoch eine '0' enthalten.

Abhilfe kann hier auf zwei Arten geschaffen werden:

1) Bei dem STEP-IN-Kommando für Laufwerk A wird im Kommandowort das 'u-Bit' gelöscht, was bedeutet, daß das Spur-Register nicht verändert wird.

2) Nach dem STEP-IN für Laufwerk A, wird das Spur-Register wieder mit dem vorherigen Wert beschrieben. In unserem Programm könnte das in der Form:

   fdc%(12)=15 : fdc%(14)=spur : call fdc#

   geschehen.

Wir haben uns für die erste Möglichkeit entschieden. Es muß allerdings für jeden STEP-IN das Kommando-Wort geändert werden, da bei Laufwerk B ein Step-IN wieder mit 'Update' erfolgen muß. Ein Schreiben des Spur-Registers bräuchte nur einmal zu erfolgen - es wäre aber ein zusätzliches 'call fdc#' nötig. Die Lösung mit dem Ändern des Kommandowortes ist, auch wenn sie zweimal ausgeführt werden muß, schneller.

Eine letzte Bemerkung - man kann es nicht oft genug wiederholen - zur Anzahl der zu übertragenden Bytes: In Lese-Richtung wird immer die gewünschte Anzahl eingegeben. Für die 9 Sektoren (à 512 Bytes) einer Spur also: 9 * $200 = $1200. Das sieht ganz logisch aus und würde wohl von jedem so gehandhabt. In Schreib-Richtung sieht die Sache etwas anders aus. Die Anzahl muß hier 9 * $200 + $20 betragen.

```
************************************************************
****             Listing: FDCCOPY.BAS                   ****
************************************************************
```

```
1 '**** Kopierprogramm für 2 Lfw. und zweiseitige Disketten A.S. ****
2 '
3 'Wir brauchen 3 Array's, für das Maschinen-Programm und als Sek-
4 'tor-Puffer für Spur-0 und Spur-1
5 '
6 dim fdc%(700),sec0%(2400),sec1%(2400):def seg=0
7 '
8 'Laden des Maschinen-Programms
9 '
10 fdc#=varptr(fdc%(0)):bload "fdcinter.img",fdc#
11 '
12 'Die Start-Adressen der beiden Puffer holen
13 '
14 sec0#=varptr(sec0%(0)):sec1#=varptr(sec1%(0))
15 'Die Anzahl zu übertragender Bytes (Lesen-$1200,Schreiben-$1220)
16 '
17 'und die Kommandoworte für STEP-IN (mit und ohne Update)
18 '
19 anzlesen=&H1200:anzschr=&H1220:stpi=&H49:stpiu=&H59
20 '
21 'Wir beginnen in jeder Spur mit Sektor-1 und POKEn Langworte
22 '
23 fdc%(15)=1:def seg=0
24 '
25 kopieren:
26 ?:fullw 2:clearw 2:gotoxy 0,1
27 ?" Kopier-Programm für zweiseitige Disketten und 2 Laufwerke"
28 ?:?:?:?" Quell-Diskette in Laufwerk A"
29 ?:?" und Ziel-Diskette in Laufwerk B einlegen."
30 ?:?:?:?" k => kopieren : andere Taste => Programm beenden"
31 if chr$(inp(2))<>"k" then end
32 '
33 init:
34 clearw 2:gotoxy 0,2
35 '
36 '------------- Lfw. B Restore und Write-Protect testen -------------
37 '
38 fdc%(12)=11:fdc%(13)=4:call fdc#
39 fdc%(12)=0:call fdc#
```

```
40 if fdc%(18) < &HA7 then goto kopi
41 '
42 ?" Diskette in Lfw.B ist schreibgeschützt! Bitte den Schreib-";
43 ?" schutz entfernen.":?
44 ?" w => weiter ; andere Taste => neu starten"
45 fdc%(12)=11:fdc%(13)=0:call fdc#
46 if chr$(inp(2))="w" then init
47 goto kopieren
48 '
49 kopi:
50 '------------------------ Lfw.A Restore --------------------------
51 fdc%(12)=11:fdc%(13)=2:call fdc#
52 fdc%(12)=0:call fdc#
53 '
54 '------------------ SPUR 0 BIS SPUR 79 KOPIEREN -------------------
55 '
56 for spur = 0 to 79 : fdc%(16)=anzlesen
57 '
58 '-------------- Seite A/0 lesen und Status anzeigen --------------
59 fdc%(12)=5:poke fdc#+56,sec0#:call fdc#
60 gotoxy 10,2:?"Spur";spur;"Seite 0 lesen "
61 gosub checkstat
62 '
63 '-------------- Seite A/1 lesen und Status anzeigen ---------------
64 fdc%(12)=11:fdc%(13)=3:call fdc#
65 fdc%(12)=5:poke fdc#+56,sec1#:call fdc#
66 gotoxy 10,2:?"Spur";spur;"Seite 1 lesen "
67 gosub checkstat
68 '
69 '------------------ Lfw.A Step-In ohne 'Update' -------------------
70 fdc%(12)=3:fdc%(4)=stpi:call fdc#
71 '
72 '-------------- Seite B/0 schreiben und Status anzeigen -----------
73 fdc%(16)=anzschr
74 fdc%(12)=11:fdc%(13)=4:call fdc#
75 fdc%(12)=6:poke fdc#+56,sec0#:call fdc#
76 gotoxy 10,4:?"Spur";spur;"Seite 0 schreiben "
77 gosub checkstat
78 '
79 '-------------- Seite B/1 schreiben und Status anzeigen -----------
```

```
80 fdc%(12)=11:fdc%(13)=5:call fdc#
81 fdc%(12)=6:poke fdc#+56,sec1#:call fdc#
82 gotoxy 10,4:?"Spur";spur;"Seite 1 schreiben "
83 gosub checkstat
84 '
85 '------------------ Lfw. B Step-In mit Update ---------------------
86 fdc%(12)=3:fdc%(4)=stpiu:call fdc#
87 '
88 '------------------ und wieder A/0 selektieren --------------------
89 fdc%(12)=11:fdc%(13)=2:call fdc#
90 '
91 next spur
92 '
93 fdc%(12)=11:fdc%(13)=0:call fdc#
94 ?:?:?"Fertig ! ......(r)estart oder (e)nde ?"
95 if chr$(inp(2))<>"r" then end
96 goto kopieren
97 '------------------------------------------------------------------
98 checkstat:
99 if fdc%(18)=&H80 and fdc%(19)=3 and fdc%(20)=0 then return
100 gotoxy 0,7:?" FDC-STATUS :$";hex$(fdc%(18))
101 ?" DMA-STATUS :$";hex$(fdc%(19))
102 ?" #DMA-BYTES :$";hex$(fdc%(21))
103 ?" TIMEOUT :$";hex$(fdc%(20)):?
104 '
105 ?" Aufgrund eines Fehlers wurde der Kopiervorgang abgebrochen."
106 ?:?" Bitte Taste drücken..."
107 fdc%(12)=11:fdc%(13)=0:call fdc#
108 key=inp(2):goto kopieren
```

### 8.3.4 Demo 3 - Erzeugung von Standard- und Fremd-Formaten

Das folgende Programm, zum beliebigen Formatieren von Disketten, soll eine weitere Anwendung der FDC-Maschinen-Routinen zeigen.

Das Programm hat einen zweifachen Nutzen. Zum einen wird hier deutlich, wie ein Spur-Puffer - der mittels WRITE-TRACK auf die Diskette geschrieben das 'Format' darstellt -

aufbereitet wird, zum anderen ist man in der Lage, Disketten so zu formatieren, daß sie auch auf anderen Computersystemen (deren Laufwerke ebenfalls durch einen WD1772 - oder kompatiblen - gesteuert werden) gelesen und beschrieben werden können. Das Gegenteil ist natürlich auch möglich. Es kann durchaus passieren, daß ein Format nicht den Anforderungen des FDC genügt und er seine Aufgabe, Sektoren in dieses Format zu übertragen, verweigert. Den folgenden Absatz sollte man sich deshalb zu Herzen nehmen.

**Wichtig!** Das Erstellen eines Formates ist eine Aufgabe, die eine genaue Kenntnis des WRITE-TRACK-Kommandos voraussetzt. Das Ändern der Parameter muß mit größter Sorgfalt erfolgen. Es funktioniert zwar vieles, aber eben nicht alles. Kurz gesagt: Falsche Werte ergeben falsche Formate. Nehmen Sie deshalb bitte die Beschreibung der FDC-Kommandos zur Hand. Dort finden Sie beschrieben, welche Änderungen die einzelnen Komponenten in der Spur, erfahren dürfen. Für die ersten 'Geh-Versuche' in der Format-Erstellung eignen sich die 'Fremd-Formate', die am Ende der WRITE-TRACK-Beschreibung aufgeführt sind.

Beschreiben wir nun aber, welche Möglichkeiten das Programm bietet.

1) Ein Spur-Puffer kann, durch zahlreiche Parameter - in weiten Grenzen - beeinflußbar, aufbereitet werden. Um überhaupt einen Überblick zu gewinnen welche Werte normalerweise Verwendung finden, sind die Parameter mit den Werten des ATARI-Formates initialisiert. Die Parameter können auch jederzeit wieder auf diese Werte zurückgesetzt werden.

   Der Puffer ist groß genug bemessen, um neben dem Spur-Format, alle eingegebenen Parameter aufzunehmen. Natürlich kann der aufbereitete Puffer

nicht komplett sein. Es gibt einige Werte, die von Spur zu Spur bzw. von Seite zu Seite verschieden sind. Z.B. wird als Spur-Angabe im ID-Feld die Spur-Nummer eingesetzt, die der zu formatierenden Spur entspricht. Während eine Diskette formatiert wird, muß eine solche Information also ständig angepaßt werden. Die dazu notwendigen Adressen der ID-Felder, werden ebenfalls im Spur-Puffer untergebracht.

2) Sie brauchen sich keine Werte zu notieren, um sie bei jedem Programm-Start neu einzugeben und dann den Puffer in gewünschter Weise aufbereiten lassen. Ein 'formatierter' Puffer kann als File auf eine Diskette geschrieben werden. Sie können sich also eine kleine 'Format-Bibliothek' mit Standard- und Kopierschutz-Formaten anlegen.

3) Wozu den Puffer-Inhalt retten, wenn er später nicht wieder geladen werden kann? Ein, auf Diskette befindliches Format-File, kann natürlich in den Spur-Puffer geladen werden.

4) Es ist wohl selbstverständlich, daß die Möglichkeit, eine Diskette zu formatieren nicht vergessen wurde. Diese Formatierungs-Routine übernimmt die Anpassung der Spur-Nummer in den ID-Feldern. Sie können noch einen 'Offset' angeben, der zur Spur-Nummer addiert wird. Dies ist im allgemeinen aber nur einem Kopierschutz dienlich.

```
*************************************************************
****                                                     ****
****              Listing: FDCFRMT.BAS                   ****
****                                                     ****
*************************************************************
```

```
10 '*******************************************************************
12 '*****        Disketten beliebig formatieren A.S. (7/86)        *****
14 '*******************************************************************
16 dim fdc%(700) :fdc# =varptr(fdc%(0))
18 dim trk%(3200):trk# =varptr(trk%(0))
20 bload "fdcinter.img",fdc#
22 def seg=0:poke fdc#+52,trk#
24 '*******************************************************************
26 gosub default:'Standard-Werte für ATARI-FORMAT lesen
28 menue:
30 ?:fullw 2: clearw 2:gotoxy 0,0:width 80
32 '
34 ?" a) GAP1 ändern                              |"
36 ?" b) GAP2 ändern                              |"
38 ?" c) GAP3 (Teil 1) ändern                     |"
40 ?" d) GAP3 (Teil 2) ändern                     |"
42 ?" e) GAP4 ändern                              |"
44 ?" f) DATEN-FELD ändern                        |"
46 ?" g) SYNC-Bytes (vor dem ID-Feld) ändern      |"
48 ?" h) SYNC-Bytes (vor dem Daten-Feld) ändern   |"
50 ?" i) DATA-ADDRESS-MARK ändern                 |"
52 ?" j) START-SEKTOR ändern                      |"
54 ?" k) SEKTOR-LÄNGE (im ID-FELD) ändern         |"
56 ?" l) RECORD-ANZAHL ändern                     |"
58 ?" m) GAP5 ändern                              |"
60 ?" ==================================================";
62 ?"========================="
64 ?" n) Spur-Puffer aufbereiten          ";
66 ?" q) Werte auf ATARI-FORMAT setzen"
68 ?" o) Spur-Puffer von Disk laden       ";
70 ?" r) Diskette formatieren"
72 ?" p) Spur-Puffer als File speichern   ";
74 ?" s) Programm beenden"
76 '
78 for prm=0 to 15 Step 2
80 gotoxy 21,prm/2:?"Anzahl:";trk%(3150+prm);" "
82 gotoxy 30,prm/2:?"Wert:$ ";hex$(trk%(3151+prm));" ":next prm
84 for prm=16 to 20
86 gotoxy 30,prm-8:?"Wert:$ ";hex$(trk%(3150+prm));" ":next prm
88 '
90 taste:
```

```
92 gotoxy 0,18:?spc(60):gotoxy 0,18:?" Welche Funktion?";
94 key=inp(2):if chr$(key)<"a" or chr$(key)>"s" then 94
96 wahl=key+1-asc("a")
98 if wahl=19 then end
100 if wahl<9 then goto zweiwerte
102 if wahl<14 then goto einwert
104 wahl=wahl-13:
106 on wahl gosub aufbereiten,laden,speichern,default,formatieren
108 goto menue
110 '
112 '================== Anzahl und Wert eingeben ======================
114 zweiwerte:
116 gotoxy 0,18:?spc(60):gotoxy 0,18
118 ?" >> ";chr$(key);" << Bitte neue Anzahl eingeben: ";:input anz$
120 if len(anz$)=0 then goto zwei
122 trk%(3148+wahl*2)=val(anz$):gotoxy 21,wahl-1:?"Anzahl:";
124 ?trk%(3148+wahl*2);" "
126 zwei:
128 gotoxy 0,18:?spc(60):gotoxy 0,18
130 ?" >> ";chr$(key);" << Bitte neuen Wert eingeben: ";:input w$
132 if len(w$)=0 then goto taste
134 trk%(3149+wahl*2)=val(w$):gotoxy 30,wahl-1:?"Wert:$ ";
136 ?hex$(trk%(3149+wahl*2));" ":goto taste
138 '
140 '====================== Wert eingeben ============================
142 einwert:
144 gotoxy 0,18:?spc(60):gotoxy 0,18
146 ?" >> ";chr$(key);" << Bitte neuen Wert eingeben: ";:input w$
148 if len(w$)=0 then goto taste
150 trk%(3157+wahl)=val(w$):gotoxy 30,wahl-1:?"Wert:$ ";
152 ?hex$(trk%(3157+wahl));" ":goto taste
154 '
156 '====================== SPUR-PUFFER AUFBEREITEN ==================
158 aufbereiten:
160 clearw 2:gotoxy 12,0:?"Spur-Puffer aufbereiten":?
162 '------------------------ GESAMTLÄNGE TESTEN ---------------------
164 gesamt=0
166 for i=3152 to 3164 step 2:gesamt=gesamt+trk%(i):next i
168 gesamt=(gesamt+9)*trk%(3169)+trk%(3150)
170 if gesamt <= 6234 then goto weiter
```

```
172 if gesamt >6250 then ?:?:?" Spur-Information zu lang":goto fail
174 ?:?" Für GAP5 (Spur-Nachspann) bleiben nur";6250-gesamt;
176 ?" Byte übrig.":?:?" Das ist zu wenig!"
178 fail:
180 ?:?" Bitte Taste drücken ...":key=inp(2):return
182 weiter:
184 '------------------------ PUFFER SCHREIBEN -------------------------
186 offset=1:?" Spur-Vorspann (";trk%(3150);"Byte )
188 anzahl=trk%(3150):wert=trk%(3151):gosub bufpoke:'  GAP1
190 for record = 1 to trk%(3169):?" Record:";record
192 anzahl=trk%(3152):wert=trk%(3153):gosub bufpoke:'  GAP2
194 anzahl=trk%(3162):wert=trk%(3163):gosub bufpoke:'  SYNC
196 def seg=offset:trk%(3170+record)=offset:'ID-Adr. merken
198 poke trk#-1,&Hfe:'                                  ID-AM
200 poke trk#+2,record-1+trk%(3167):'                   START-SEKTOR
202 poke trk#+3,trk%(3168):'                            SEKTOR-LÄNGE
204 poke trk#+4,&Hf7:offset=offset+6:'                  ID-CRC
206 anzahl=trk%(3154):wert=trk%(3155):gosub bufpoke:'  GAP3
208 anzahl=trk%(3156):wert=trk%(3157):gosub bufpoke:'  GAP3
210 anzahl=trk%(3164):wert=trk%(3165):gosub bufpoke:'  SYNC
212 def seg=offset:poke trk#-1,trk%(3166):'             DAM
214 offset=offset+1
216 anzahl=trk%(3160):wert=trk%(3161):gosub bufpoke:'  SEKTOR-DATEN
218 def seg=offset:poke trk#-1,&Hf7:offset=offset+1:'  DATA-CRC
220 anzahl=trk%(3158):wert=trk%(3159):gosub bufpoke:'  GAP4
222 next record
224 ?" Spur-Nachspann (";6250-offset;"Byte )"
226 anzahl=6300-offset:wert=trk%(3170):gosub bufpoke:'  GAP5
228 '
230 ?:?:?" Puffer ist aufbereitet!":pready=1:return
232 '-------------------------------------------------------------------
234 bufpoke:
236 For i = 0 to anzahl-1:def seg=offset+i:poke trk#-1,wert:next i
238 offset=offset+anzahl:return
240 '
242 '================== STANDARD-WERTE FÜR ATARI-FORMAT ================
244 default:
246 restore 406:for i=3150 to 3170:read standard:trk%(i)=standard
248 next i: pready=0:return
250 '
```

```
252 '================== SPUR-PUFFER AUF DISK SCHREIBEN =================
254 speichern:
256 clearw 2:gotoxy 12,2:?"Spur-Puffer auf Disk schreiben":?:?:?
258 if pready=1 then goto speich2
260 ?" Spur-Puffer ist noch leer. Vor dem Speichern zuerst Puffer ";
262 ?"aufbereiten!":?:?" Bitte Taste drücken ...":key=inp(2):return
264 speich2:
266 ?:?:?" Puffer-Daten als Disk-File speichern (j/n) ?":?:?
268 if chr$(inp(2))<>"j" then return
270 input " Bitte File-Namen für die Format-Daten eingeben :",file$
272 bsave file$,trk#,6402:return
274 '
276 '================== SPUR-PUFFER VON DISK LADEN ===================
278 laden:
280 clearw 2:gotoxy 12,2:?"Spur-Puffer von Disk laden":?:?:?
282 ? " Unter welchem Programm-Namen ist das fertige Format abge";
284 ? "speichert? ":?:input file$
286 open"R",1,file$,1:test=lof(1):close 1
288 if test=6402 then bload file$,trk#:pready=1:return
290 ?:?:? " Dies ist kein Datenfile !!! Bitte Taste drücken ...."
292 a=inp(2):return
294 '======================== DISKETTE FORMATIEREN =====================
296 formatieren:
298 clearw 2:gotoxy 12,2:?"Diskette formatieren":?:?:?
300 if pready=1 then goto frmt2
302 ?" Spur-Puffer ist noch leer. Bitte zuerst den Spur-Puffer ";
304 ?"aufbereiten!":?:?" Bitte Taste drücken ...":key=inp(2):return
306 frmt2:
308 ?" Soll eine Diskette formatiert werden (j/n) ?"
310 if chr$(inp(2))<>"j" then return
312 '
314 ?:?:?" Bitte zu formatierende Diskette in Lfw. A einlegen.":?
316 input " Ab welcher Spur ?",a$:a=val(a$):if a<0 or a>82 then 316
318 input " Bis zu welcher Spur ?",a$:b=val(a$)
320 if b<0 or b>82 then 318:if b<a then 316
322 ?:?" Welchen Offset sollen die Spur-Nummern in den ID-Feldern ";
324 ?"erhalten ?":?" Es ist ein Wert von 0 bis";244-b;" möglich."
326 ?" Normalerweise ist der Offset jedoch >> 0 <<":?
328 input " Offset? ",a$:if val(a$)<0 or val(a$)>244-b then 328
330 off=val(a$)
```

```
332 ?:?" Welche Seite soll formatiert werden ?"
334 ?:?" (0)=Vorderseite oder (1)=Rückseite"
336 key=inp(2)
338 if chr$(key)="0" then drive=2:goto format
340 if chr$(key)="1" then drive=3:goto format
342 goto 336
344 format:
346 ?:?" Letzte Möglichkeit zum Abbruch! (f)ormatieren (q)uit"
348 key=inp(2):if chr$(key)="q" then goto formatieren
350 if chr$(key)<>"f" then 348
352 '--------------------- formatieren ----------------------
354 fdc%(12)=11:fdc%(13)=drive:call fdc#:' Laufwerk selektieren
356 nochmal:
358 fdc%(12)=0:call fdc#:'RESTORE
360 if fdc%(18)<&Ha7 then goto sek
362 ?:?" *** FEHLER! Die Diskette ist schreibgeschützt!"
364 ?:?" (w)iederholen oder (q)uit ?"
366 key=inp(2):if chr$(key)="q" then goto frmt
368 if chr$(key)="f" then fdc%(12)=10:call fdc#:goto nochmal
370 goto 366
372 sek:
374 fdc%(12)=1:fdc%(14)=a:call fdc# :' S/L-Kopf auf Start-Spur
376 '
378 fdc%(16)=6400
380 for spur=a to b
382 for record=1 to trk%(3169)
384 def seg=trk%(3170+record):poke trk#,spur+off
386 poke trk#+1,drive-2:next record
388 '
390 fdc%(12)=8:call fdc# :' WRITE-TRACK
392 if fdc%(18)<>&H80 or fdc%(19)<>3 then ?" *** FEHLER! Spur:";spur
394 fdc%(12)=3:call fdc# :' STEP-IN
396 next spur
398 frmt:
400 fdc%(12)=11:fdc%(13)=0:call fdc#:goto formatieren:'deselektieren
402 end
404 '------------------- Werte für ATARI-FORMAT ----------------------
406 data 60,78,12,0,22,78,12,0,40,78,512,229,3,245,3,245,251,1,2,9,78
```

### 8.3.5 Demo 4 - Konvertieren von Ein- nach Zweiseitig

Als letztes Beispiel zur Anwendung der FDC-Maschinenroutinen, folgt nun ein, für Besitzer von zweiseitigen Laufwerken, überaus nützliches Programm.

Wenn Sie eine einseitig formatierte Diskette kopieren möchten, so stellt das natürlich kein Problem dar. Aber wenn Sie versuchen, diese auf eine zweiseitig formatierte Diskette zu kopieren, wird ihnen das DESK-TOP eine Fehlermeldung ausgeben. Sinngemäß bedeutet diese Meldung, daß Quell- und Ziel-Diskette nicht das gleiche Format besitzen und ein Kopieren aus diesem Grund nicht möglich ist. Als Besitzer von zweiseitigen Laufwerken sind Sie natürlich bestrebt, die doppelte Speicherkapazität auszunutzen. In diesem Fall bleibt ihnen nur die Möglichkeit, alle Dateien einzeln auf die zweiseitige Diskette zu kopieren.

Solange sich nur einige Dateien auf der Quell-Diskette befinden, ist diese Form des Kopierens sogar schneller als ein komplettes Kopieren der Diskette. Zeitintensiv wird die Sache allerdings, wenn sehr viele Dateien kopiert werden müssen. Es ist wirklich keine Seltenheit, daß sich auf solchen Disketten fünfzig (oder noch mehr) Dateien befinden. Der Zeitbedarf für eine derartige Kopier-Session ist schon beträchtlich. Aber Sie werden schon richtig vermuten: Die bei solchen Gelegenheiten immer wieder zitierte 'Tasse Kaffee' läßt sich umgehen.

Wie schon erwähnt wurde, ist das DESK-TOP nicht in der Lage, die gesamte Diskette zu kopieren, wenn nicht Quell- und Ziel-Diskette das gleiche Format besitzen. Das es nicht möglich sein kann, eine zweiseitige auf eine einseitige Diskette zu kopieren, ist ja zu verstehen. Denn die 720 kB einer zweiseitigen Diskette, wird man auf einer einseitigen Diskette (360 kB) wohl kaum unterbringen können. Im umgekehrten Fall (der hier beschrieben wird) paßt die gesamte Information einer einseitigen Diskette jedoch bequem auf eine zweiseitige Diskette. Speicherplatzprobleme werden hier also nicht auftreten.

Bei diesen Betrachtungen darf man allerdings nicht vergessen, daß auf jeder Diskette Bereiche reserviert sind, in denen sich

wichtige Informationen befinden (Directory, FAT, Boot-Sektor). Eine genaue Erklärung dazu finden Sie in Kapitel 3. Es muß sichergestellt sein, daß sich diese Informationen nach dem Kopieren auch an der Stelle befinden, wo das Betriebssystem sie erwartet.

Teilt man eine Diskette, so wie es das Betriebssystem macht, in 'logische Sektoren' ein, so besteht zwischen ein- und zweiseitigen Disketten lediglich ein Unterschied in deren Anzahl. Während eine einseitige Diskette in 720 logische Sektoren eingeteilt ist, befinden sich auf einer zweiseitigen Diskette 1440 logische Sektoren. Glücklicherweise sind die logischen Sektor-Adressen für FAT, Directory und Boot-Sektor für beide Formate identisch. Der reservierte Platz für diese Komponenten besitzt ebenfalls die gleiche Größe.

Die Aufgabe des Programms reduziert sich somit auf das Umrechnen von logischen, in physikalische Sektor-Adressen. Dazu ein kleines Beispiel:

Die Numerierung der logischen Sektoren beginnt mit '0'. Das Directory beginnt im logischen Sektor 11, also dem 12. Sektor auf der Diskette. Doch wo ist auf einer Diskette der 12. Sektor? Genau diese Information enthält die physikalische Sektor-Adresse. Auf einer einseitigen Diskette befindet sich der logische Sektor 11 auf Seite 0/Spur 1/Sektor 3. Auf der zweiseitigen Diskette ist dieser Sektor jedoch auf Seite 1/Spur 0/Sektor 3 zu finden.

Nach soviel Vorrede kommen wir jetzt endlich zu dem Programm. Die wichtigsten Schritte im Programmablauf sollen aber zuvor noch erläutert werden.

1. Die einseitige Quell-Diskette wird in Lfw. A und die zweiseitig formatierte Ziel-Diskette in Lfw. B eingelegt.

2. Der Boot-Sektor der Ziel-Diskette wird 'gerettet'.

3. Es werden nun jeweils zwei aufeinanderfolgende Spuren der Quell-Diskette gelesen und auf eine Spur (Vorder-/Rückseite) der Ziel-Diskette geschrieben, bis achtzig Spuren kopiert sind.

4. Der Boot-Sektor wird wieder zurückgeschrieben.

Nach diesem Konvertieren befinden sich alle Programme der einseitigen Diskette auf der zweiseitigen Diskette.

```
10 '** Konvertieren von Ein- nach Zweiseitig A.S. 10/86 **
20 '
30 'Laden des Maschinen-Programms
40 dim fdc%(700) : fdc#=varptr(fdc%(0)) : bload "fdcinter.img",fdc#
50 '
60 'Wir benötigen 3 Puffer und deren Start-Adressen
70 dim sec0%(2400) : sec0#=varptr(sec0%(0))
80 dim sec1%(2400) : sec1#=varptr(sec1%(0))
90 dim bootsec%(512) : bootsec#=varptr(bootsec%(0))
100 '
110 def seg=0 :' Wir POKEn Langworte
120 '
130 anzlesen=&H1200 :' READ_SECTOR => 9 * 512 Byte
140 anzschr =&H1220 :' WRITE_SECTOR => 9 * 512 + 32 Byte
150 '
160 fdc%(15)=1 :' Wir beginnen in jeder Spur mit Sektor 1
170 '
180 '*** START ****************************************************
190 '
200 start:
210 ? : fullw 2 : clearw 2 : gotoxy 0,1
220 ? " Konvertieren von Einseitigen in zweiseitige Disketten"
230 ?:?:?:? " Einseitige Quell-Diskette in Laufwerk A"
240 ?:? " und zweiseitige Ziel-Diskette in Laufwerk B einlegen."
250 ?:?:?:? " k => konvertieren : andere Taste => Programm beenden"
260 if chr$(inp(2))<>"k" then end
270 '
280 init:
290 clearw 2 : gotoxy 0,2
```

```
300 '
310 '*** Lfw. B => Restore und Write-Protect testen ****************
320 '
330 fdc%(12)=11 : fdc%(13)=4 : call fdc#
340 fdc%(12)=0 : call fdc#
350 if fdc%(18) < &HA7 then goto convert
360 '
370 ? " Diskette in Lfw.B ist schreibgeschützt! Bitte den Schreib-";
380 ? " schutz entfernen.":?
390 ? " w => weiter ; andere Taste => neu starten"
400 fdc%(12)=11 : fdc%(13)=0 : call fdc#
410 if chr$(inp(2))="w" then init
420 goto start
430 '
440 convert:
450 ?" Konvertierung läuft. Bitte etwas Geduld .....
460 '
470 '*** Lfw.B => Bootsektor lesen ***********************************
480 '
490 fdc%(16)=&H200 : fdc%(12)=5 : poke fdc#+56,bootsec# : call fdc#
500 gosub checkstat
510 '
520 '*** Lfw.A => Restore ********************************************
530 '
540 fdc%(12)=11 : fdc%(13)=2 : call fdc#
550 fdc%(12)=0 : call fdc#
560 '
570 '*** Spur 0-79 (Lfw.A) nach Spur 0-39 (Lfw.B) kopieren *********
580 '
590 trackcnta=0 : trackcntb=0 :' Spurzähler für Lfw.A und Lfw.B
600 loop:
610 '
620 '*** Lfw.A => Spur X lesen (X=trackcnta) ***********************
630 '
640 fdc%(12)=11 : fdc%(13)=2 : call fdc# :' Lfw.A/0 selektieren
650 fdc%(12)=15 : fdc%(14)=trackcnta : call fdc# :' Spurreg. schr.
660 fdc%(16)=anzlesen
670 fdc%(12)=5 : poke fdc#+56,sec0# : call fdc# :' READ_SECTOR
680 gosub checkstat
690 fdc%(12)=3 : call fdc# :' STEP_IN
```

```
700 '
710 '*** Lfw.A => Spur X+1 lesen **********************************
720 '
730 fdc%(12)=5 : poke fdc#+56,sec1# : call fdc# :' READ_SECTOR
740 gosub checkstat
750 fdc%(12)=3 : call fdc# :' STEP_IN
760 trackcnta=trackcnta+2
770 '
780 '*** Lfw.B => Spur X auf Seite 0 schreiben ********************
790 '
800 fdc%(12)=11 : fdc%(13)=4 : call fdc# :' Lfw. B/0 selektieren
810 fdc%(12)=15 : fdc%(14)=trackcntb : call fdc# :' Spurreg. schr.
820 fdc%(16)=anzschr
830 fdc%(12)=6 : poke fdc#+56,sec0# : call fdc# :' WRITE_SECTOR
840 gosub checkstat
850 '
860 '*** Lfw.B => Spur x+1 auf Seite 1 schreiben ******************
870 '
880 fdc%(12)=11 : fdc%(13)=5 : call fdc# :' Lfw. B/1 selektieren
890 fdc%(12)=6 : poke fdc#+56,sec1# : call fdc# :' WRITE_SECTOR
900 gosub checkstat
910 fdc%(12)=3 : call fdc# :' STEP_IN
920 trackcntb=trackcntb+1
930 '
940 '*** Testen, ob schon 80 Spuren geschrieben wurden *************
950 '
960 if trackcntb<40 then goto loop
970 '
980 '*** Lfw.B => Restore und Boot-Sektor zurückschreiben **********
990 '
1000 fdc%(12)=11 : fdc%(13)=4 : call fdc# :' Lfw. B/0 selektieren
1010 fdc%(12)=0 : call fdc# :' RESTORE
1020 fdc%(16)=&H220
1030 fdc%(12)=6 : poke fdc#+56,bootsec# : call fdc# :' WRITE_SECTOR
1040 gosub checkstat
1050 '
1060 '*** Lfw.A => Restore und deselektieren **********************
1070 '
1080 fdc%(12)=11 : fdc%(13)=2 : call fdc# :' Lfw. A/0 selektieren
1090 fdc%(12)=0 : call fdc# :' RESTORE
```

```
1100 fdc%(12)=11 : fdc%(13)=0 : call fdc# :' deselektieren
1110 '
1120 '*** FERTIG ****************************************************
1130 '
1140 ?:?:? " Fertig ! ......(r)estart oder (e)nde ?"
1150 if chr$(inp(2))<>"r" then end
1160 goto start
1170 '
1180 '*** Status testen ***********************************************
1190 '
1200 checkstat:
1210 if fdc%(18)=&H80 and fdc%(19)=3 and fdc%(20)=0 then return
1220 gotoxy 0,7 : ? " FDC-STATUS :$";hex$(fdc%(18))
1230 ? " DMA-STATUS :$";hex$(fdc%(19))
1240 ? " #DMA-BYTES :$";hex$(fdc%(21))
1250 ? " TIMEOUT :$";hex$(fdc%(20)):?
1260 '
1270 ? " Aufgrund eines Fehlers wurde das Konvertieren abgebrochen."
1280 ?:? " Bitte Taste drücken..."
1290 fdc%(12)=11 : fdc%(13)=0 : call fdc# :' Deselektieren
1300 key=inp(2) : goto start
1310 '
1320 '****************************** ENDE ******************************
```

## 8.4 Erstellung von BASIC-Ladern

Dieses Kapitel hat nur indirekt mit der Floppy zu tun. Es handelt sich hier vielmehr um ein einfaches, aber dennoch sehr nützliches 'Tool' für welches wohl viele eine Verwendung haben. Es sollen sich dazu die Assembler-Freaks, die für alles die passende Maschinensprache-Lösung parat haben, in besonderem Maße angesprochen fühlen. Es muß nicht immer ein komplettes 'Public-Domain-Programm' sein, welches man der Allgemeinheit zur Verfügung stellt. Oft kann man schon mit 'Kleinigkeiten' anderen eine Freude machen. Aber lesen Sie selbst.

Es lassen sich viele Programmierprobleme - recht komfortabel und schnell - durch ein BASIC-Programm lösen. Die Grenzen dieser Sprache sind jedoch dann erreicht, wenn es auf höchste

Geschwindigkeit ankommt. Es sind meist nur Kleinigkeiten, die einen daran hindern, eine gute Programmidee in BASIC zu verwirklichen. Wenn es z.B darum geht, eine bewegte Grafik für ein Spielprogramm zu erzeugen, muß der reine BASIC-Programmierer passen. Sicher - auch eine umfangreiche Dateiverwaltung läßt sich vollständig in BASIC schreiben. Aber wehe, wenn es an das Sortieren großer Datenmengen geht.

Doch kommen wir zum eigentlichen Punkt. Es gibt Assemblerprogrammierer unter uns, die nützliche kleine Maschinen-Routinen am Fließband produzieren. Oft erstellen dieselben Leute ebenfalls BASIC-Programme. Die Geschwindigkeitsprobleme des reinen BASIC-Programmierers scheinen für sie nicht zu existieren. Es ist ja auch ganz klar; die Programmstellen für die sich der BASIC-INTERPRETER zu viel Zeit läßt, werden flugs durch eine Maschinen-Routine ersetzt. Die Einbindung dieser ist ja ausgesprochen einfach. Ein simples 'call ...' wird später dann die vorherigen Probleme beseitigen.

Es wäre schön, wenn diese Routinen auch denen zur Verfügung ständen, die keinen ASSEMBLER besitzen. Es liegt selten daran, daß die Maschinen-Routine nicht preisgegeben werden soll. Aber - es muß ein BASIC-Programm geschrieben werden, welches die Daten der Maschinen-Routine enthält und diese entweder in den Speicher POKEt oder einen Datenfile (zwecks späteren 'bload') auf der Diskette erzeugt. Aber weshalb sollte sich jemand 'für andere' eine solche Arbeit machen? Für den eigenen Gebrauch ist es schließlich nicht nötig. Im Besitz des Maschinenfiles ist man ja bereits - der ASSEMBLER hat diese Aufgabe schon erledigt.

Wenn aber ein Programm - um ein solches geht es hier - die Erstellung des Datenfiles übernimmt, ist es dann noch ein großer Aufwand, den reinen BASIC-Programmierern die Unter-Routinen zur Verfügung zu stellen? Wir sind überzeugt, das diese für eine 'ready-to-hack-in' Lösung dankbar sein werden.

Das BASIC-Programm erledigt die Aufgabe einer ASSEMBLER-BASIC-ASSEMBLER-Konvertierung. Es wird nach Programm-

start der Name des zu konvertierenden Files erwartet. Danach müssen noch zwei weitere Filenamen eingegeben werden.

1. Der Name des zu erzeugenden BASIC-Laders. Dieses Programm wird wohl in den meisten Fällen, in Form eines Listings, an andere weitergegeben. Da bei der Eingabe, besonders wenn viele DATA-Zeilen enthalten sind, schon mal Fehler auftreten, ist eine Prüfsumme hierin enthalten.

2. Der Name des Files, den der BASIC-Lader später erzeugen soll. Der Lader testet, ob die Prüfsumme der Summe aller DATAs entspricht. Ist das der Fall, so wird ein File auf die Diskette geschrieben, der mit dem ursprünglich konvertierten, identisch ist.

```
**************************************************************************
****                                                                  ****
****                    Listing: DATAMAKE.BAS                         ****
****                                                                  ****
**************************************************************************

10 '**********************************************************************
20 '**************          Data-Maker A.S. 10/86          ***************
30 '**********************************************************************
40 '
50 'Erzeugt aus einem beliebigen File ein BASIC-Programm. Wird
60 'dieses später mit 'RUN' gestartet, so wird ein File geschrie-
70 'ben, das mit dem zuvor übergebenen identisch ist.
80 '
90 '**********************************************************************
100 '
110 ?:fullw 2:clearw 2:gotoxy 0,0
120 input "Aus welchem File sollen DATA's erzeugt werden "; prg$
130 '
140 open"R",#1,prg$,1:bytes=lof(1):close: feldlen=cint(bytes/2-1)
150 ?:?"INFO --> Die Länge von >> ";prg$;" << beträgt ";bytes;" Byte"
160 ?:?:?
170 '
```

```
180 input "Wie soll das Ausgabe-File heißen ";bas$:?:?
190 '
200 ?"In das File >> ";bas$;" << wird ein Lader integriert, der"
210 ?:?"aus den DATA's ein File erstellt, das dem"
220 ?:?"Eingabe-File >> ";prg$;" << entspricht.":?:?:?
230 input "Bitte auch hierfür einen Namen eingeben ";make$
240 ?:?"***********************************************************"
250 ?:?"Jetzt gehts los! Gegen Langeweile gibt's einige Informationen"
260 ?"zu den einzelnen Vorgängen"
270 ?:?:?"Ein Integer-Array (c%) wird mit ";feldlen;" dimensioniert"
280 dim c%(feldlen)
290 ?
300 ?"Das Eingabe-File >> ";prg$;" << wird nach varptr(c%(0)) geladen"
310 bload prg$,varptr(c%(0))
320 '
330 ?:?"Das Ausgabe-File >> ";bas$;" << wird geöffnet"
340 open"O",1,bas$
350 '
360 ?:?:?"Der Inhalt des c%-Array's wird nun, in Form von Data-Zeilen,"
370 ?"in dieses File geschrieben."
380 ?:?:?"Bitte etwas Geduld ......"
390 '
400 check#=0:z=0:zl=100
410 '
420 if z mod 8 =0 then print #1:print #1,str$(zl);" DATA ";:zl=zl+1
430 '
440 print #1,right$("0000"+hex$(c%(z)),4);
450 '
460 check#=check#+c%(z):z=z+1
470 if z=feldlen+1 then 510
480 if z mod 8 <> 0 then print #1,",";
490 goto 420
500 '
510 ?:?:?"So, jetzt nur noch das Lade-Programm anhängen ...."
520 '
530 print #1
540 print #1,str$(10);" '********* File-Maker A.S. *********"
550 print #1,str$(15);" '"
560 print #1,str$(20);" ?:fullw 2:clearw 2:gotoxy 0,0"
570 print #1,str$(25);" ? ";chr$(34);"File >> ";make$;
```

```
580 print #1," << wird erzeugt";chr$(34);":?:?:?"
590 '
600 print #1,str$(30);" dim c%(";str$(feldlen);"):cs#=0"
610 '
620 print #1,str$(35);" for i=0 to ";str$(feldlen)
630 '
640 print #1,str$(40);" read a$:c%(i)=val(";chr$(34);"&H";chr$(34);"+a$)
"
650 '
660 print #1,str$(45);" check#=check#+(c%(i))"
670 '
680 print #1,str$(50);" next i"
690 '
700 print #1,str$(55);" if check#=";str$(check#);" then ";str$(70)
710 '
720 print #1,str$(60);" ?";chr$(34);"Geht leider noch nicht, ";
730 print #1,"da etwas mit den DATA's nicht stimmt.";chr$(34)
740 print #1,str$(65);" goto 80"
750 '
760 print #1,str$(70);" bsave ";chr$(34);make$;chr$(34);",varptr(c%(0)),
";
770 print #1,str$(bytes)
780 '
790 print #1,str$(75);" ? ";chr$(34);"Das Programm >> ";make$;
800 print #1," << ist nun geschrieben.";chr$(34)
810 '
820 print #1,str$(80);" ?:?:?:?";chr$(34);"Bitte Taste drücken";chr$(34)
;
830 print #1,":a=inp(2):end"
840 print #1,str$(85);" '"
850 print #1,str$(90);" '********* DATA's für ";make$;" **********"
860 print #1,str$(95);" '"
870 '
880 ?:?:?"...das Ausgabe-File schließen und ...
890 close #1
900 '
910 ?:?"das Programm >> ";bas$;" >> ist fertig."
920 ?:?"Bitte Taste drücken":a=inp(2):end
```

# Anhang I - File-Maker für editor.tos

```
10   '*********    File-Maker      A.S.    *********
15   '
20   ?:fullw 2:clearw 2:gotoxy 0,0
25   ? "File >> editor << wird erzeugt":?:?:?
30   dim c%( 8444):cs#=0
35   for i=0 to  8444
40   read a$:c%(i)=val("&H"+a$)
45   check#=check#+(c%(i))
50   next i
55   if check#= 81578547.2 then 70
60   ?"Geht leider noch nicht, da etwas mit den DATA's nicht stimmt."
65   goto 80
70   bsave "editor.tos",varptr(c%(0)), 16890
75   ? "Das Programm >> editor << ist nun geschrieben."
80   ?:?:?:?"Bitte Taste drücken":a=inp(2):end
85   '
90   '********* DATA's für editor **********
95   '
100  DATA 601A,0000,2FF6,0000,0DDC,0000,9D28,0000
101  DATA 0000,0000,0000,0000,0000,0000,2A4F,2A6D
102  DATA 0004,202D,000C,D0AD,0014,D0AD,001C,D0BC
103  DATA 0000,1100,220D,D280,C2BC,FFFF,FFFE,2E41
104  DATA 2F00,2F0D,3F00,3F3C,004A,4E41,DFFC,0000
105  DATA 000C,3F3C,0003,3F3C,000B,3F3C,0023,4E4E
106  DATA 5C8F,6108,2F3C,0000,0000,4E41,6100,2A1A
107  DATA 6100,2EAC,6100,2DD6,610A,6100,01F4,6100
108  DATA 00EA,4E75,6100,2E08,6100,2DC2,33FC,0000
109  DATA 0000,3DE2,33FC,0000,0000,3DE6,33FC,0000
110  DATA 0000,3DE8,33FC,0001,0000,3DE4,303C,0000
111  DATA 33FC,0006,0000,3DF0,33FC,0001,0000,3DF2
112  DATA 33FC,004F,0000,3DEC,33FC,0009,0000,3DEE
113  DATA 33FC,0009,0000,3E5A,13FC,0030,0000,32C1
114  DATA 13FC,0039,0000,32C2,33FC,05DC,0000,3DF4
115  DATA 23FC,0000,AC1A,0000,3E60,6102,4E75,6100
116  DATA 2E1E,33FC,0014,0000,3E02,33FC,000A,0000
117  DATA 3E04,6100,2DAE,207C,0000,3083,6100,293C
118  DATA 33FC,0014,0000,3E02,33FC,000C,0000,3E04
```

```
119  DATA 6100,2D90,207C,0000,30B5,6100,291E,33FC
120  DATA 0014,0000,3E02,33FC,000E,0000,3E04,6100
121  DATA 2D72,207C,0000,30E7,6100,2900,6100,2DF2
122  DATA 6100,2CEA,6100,2DB8,4E75,6100,2D90,4A80
123  DATA 67F8,4840,B03C,0044,672A,B03C,004B,6604
124  DATA 6140,60E6,B03C,004D,6604,6158,60DC,B03C
125  DATA 0050,6604,611E,60D2,B03C,0048,6602,6106
126  DATA 60C8,508F,4E75,23F9,0000,3E18,0000,3DDE
127  DATA 615A,4E75,23F9,0000,3E1C,0000,3DDE,614C
128  DATA 4E75,2039,0000,3DDA,5380,6708,23C0,0000
129  DATA 3DDA,600A,23F9,0000,3DD6,0000,3DDA,6100
130  DATA 2A3C,4E75,2039,0000,3DDA,5280,B0B9,0000
131  DATA 3DD6,6E08,23C0,0000,3DDA,600A,23FC,0000
132  DATA 0001,0000,3DDA,6100,2A14,4E75,6100,2D10
133  DATA 2079,0000,3DDE,2039,0000,3DDA,5380,E588
134  DATA 2270,0800,4ED1,33FC,000A,0000,3E02,33FC
135  DATA 0002,0000,3E04,6100,2C8A,6100,2C04,302F
136  DATA 0004,4440,B07C,001D,6D04,303C,001D,E548
137  DATA 227C,0000,3BC4,2071,0000,6100,27FE,6100
138  DATA 2CF0,6100,2BDC,6100,2C46,205F,548F,4ED0
139  DATA 6100,2BDA,23FC,0000,0007,0000,3DD6,23FC
140  DATA 0000,0001,0000,3DDA,23FC,0000,3012,0000
141  DATA 3DD2,23FC,0000,2FF6,0000,3E18,23FC,0000
142  DATA 2FF6,0000,3E1C,6100,2974,4E75,6100,2B9E
143  DATA 23FC,0000,3296,0000,3DD2,23FC,0000,0008
144  DATA 0000,3DD6,23FC,0000,0005,0000,3DDA,23FC
145  DATA 0000,3256,0000,3E18,23FC,0000,3276,0000
146  DATA 3E1C,6100,2938,6100,2BB2,207C,0000,32E7
147  DATA 6100,2768,4E75,23FC,0000,0006,0000,3DD6
148  DATA 23FC,0000,0004,0000,3DDA,23FC,0000,3354
149  DATA 0000,3E18,23FC,0000,336C,0000,3E1C,23FC
150  DATA 0000,3384,0000,3DD2,6100,28F2,6100,2B6C
151  DATA 207C,0000,32FA,6100,2722,4E75,6100,2B0E
152  DATA 23FC,0000,315A,0000,3DD2,23FC,0000,311A
153  DATA 0000,3E18,23FC,0000,313A,0000,3E1C,23FC
154  DATA 0000,AC1A,0000,3E60,23FC,0000,0008,0000
155  DATA 3DD6,23FC,0000,0005,0000,3DDA,6100,289E
156  DATA 6100,2B18,207C,0000,321B,6100,26CE,4E75
157  DATA 6100,092A,6100,098C,23FC,0000,0008,0000
158  DATA 3DD6,23FC,0000,0003,0000,3DDA,23FC,0000
```

```
159  DATA 33FC,0000,3DD2,23FC,0000,33BC,0000,3E18
160  DATA 23FC,0000,33DC,0000,3E1C,6100,2ACE,207C
161  DATA 0000,3449,6100,2684,6100,2842,4E75,6100
162  DATA 2A6C,23FC,0000,3786,0000,3DD2,23FC,0000
163  DATA 0008,0000,3DD6,23FC,0000,0003,0000,3DDA
164  DATA 23FC,0000,3746,0000,3E18,23FC,0000,3766
165  DATA 0000,3E1C,6100,2806,6100,2A80,207C,0000
166  DATA 3AD0,6100,2636,4E75,6100,2A22,23FC,0000
167  DATA 3A64,0000,3DD2,23FC,0000,0007,0000,3DD6
168  DATA 23FC,0000,0001,0000,3DDA,23FC,0000,3A2C
169  DATA 0000,3E18,23FC,0000,3A48,0000,3E1C,6100
170  DATA 27BC,6100,2A36,207C,0000,3AEA,6100,25EC
171  DATA 4E75,23FC,0000,0006,0000,3DD6,23FC,0000
172  DATA 0004,0000,3DDA,23FC,0000,38C6,0000,3E18
173  DATA 23FC,0000,38DE,0000,3E1C,23FC,0000,38F6
174  DATA 0000,3DD2,6100,2776,6100,29F0,207C,0000
175  DATA 3954,6100,25A6,4E75,3039,0000,3DE8,B079
176  DATA 0000,3DF0,6D06,303C,0000,6002,5240,33C0
177  DATA 0000,3DE8,D03C,0030,13C0,0000,3182,6100
178  DATA 273C,4E75,3039,0000,3DE8,B07C,0000,6F04
179  DATA 5340,6006,3039,0000,3DF0,33C0,0000,3DE8
180  DATA D03C,0030,13C0,0000,3182,6100,2710,4E75
181  DATA 3039,0000,3DE6,B07C,0001,6D06,303C,0000
182  DATA 6004,303C,0001,33C0,0000,3DE6,D03C,0030
183  DATA 13C0,0000,318C,6100,26E4,4E75,3039,0000
184  DATA 3DE6,B07C,0000,6F06,303C,0000,6004,303C
185  DATA 0001,33C0,0000,3DE6,D03C,0030,13C0,0000
186  DATA 318C,6100,26B8,4E75,3039,0000,3DE2,B079
187  DATA 0000,3DEC,6D06,303C,0000,6002,5240,33C0
188  DATA 0000,3DE2,48C0,80FC,000A,D03C,0030,13C0
189  DATA 0000,3197,4840,D03C,0030,13C0,0000,3198
190  DATA 6100,267A,4E75,3039,0000,3DE2,B07C,0000
191  DATA 6F04,5340,6006,3039,0000,3DEC,33C0,0000
192  DATA 3DE2,48C0,80FC,000A,D03C,0030,13C0,0000
193  DATA 3197,4840,D03C,0030,13C0,0000,3198,6100
194  DATA 263C,4E75,3039,0000,3DE4,B079,0000,3DEE
195  DATA 6D06,303C,0000,6002,5240,33C0,0000,3DE4
196  DATA 48C0,80FC,000A,D03C,0030,13C0,0000,31A4
197  DATA 4840,D03C,0030,13C0,0000,31A5,6100,25FE
198  DATA 4E75,3039,0000,3DE4,B07C,0000,6F04,5340
```

```
199  DATA 6006,3039,0000,3DEE,33C0,0000,3DE4,48C0
200  DATA 80FC,000A,D03C,0030,13C0,0000,31A4,4840
201  DATA D03C,0030,13C0,0000,31A5,6100,25C0,4E75
202  DATA 3039,0000,3B48,323C,0001,B07C,0400,6604
203  DATA 323C,0002,3F01,3F39,0000,3DE6,3F39,0000
204  DATA 3DE2,3F39,0000,3DE4,3F39,0000,3DE8,42A7
205  DATA 2F3C,0000,AC1A,3F3C,0008,4E4E,DFFC,0000
206  DATA 0014,4A40,6B04,6118,4E75,3F00,6100,FB78
207  DATA 6100,27E8,207C,0000,321B,6100,239E,4E75
208  DATA 33FC,0000,0000,3E08,23F9,0000,3E60,0000
209  DATA 3DF6,33FC,001F,0000,3E14,33FC,0012,0000
210  DATA 3E12,33FC,0000,0000,3EA4,33FC,00D0,0000
211  DATA 3EA6,3039,0000,3B48,B07C,0400,6618,33FC
212  DATA 0200,0000,3EA4,33FC,02D0,0000,3EA6,33FC
213  DATA 003F,0000,3E14,6102,4E75,6100,2792,6100
214  DATA 27FE,33FC,0000,0000,3E08,33F9,0000,3E08
215  DATA 0000,3E06,6100,2548,6100,27E4,6100,2770
216  DATA 6100,27BA,4840,B03C,0019,6700,00AA,B03C
217  DATA 0048,6724,B03C,0050,675C,B03C,001C,6712
218  DATA B03C,004B,670C,B03C,004D,66D4,6100,FA76
219  DATA 6004,6100,FA4E,4E75,3039,0000,3E08,B07C
220  DATA 0000,672E,B079,0000,3EA6,6714,0479,0100
221  DATA 0000,3E08,04B9,0000,0100,0000,3DF6,6012
222  DATA 0479,00D0,0000,3E08,04B9,0000,00D0,0000
223  DATA 3DF6,6000,FF76,3039,0000,3E08,B079,0000
224  DATA 3EA6,672E,B079,0000,3EA4,6614,0679,00D0
225  DATA 0000,3E08,06B9,0000,00D0,0000,3DF6,6012
226  DATA 0679,0100,0000,3E08,06B9,0000,0100,0000
227  DATA 3DF6,6000,FF36,33FC,0000,0000,3A1C,48F9
228  DATA 38F8,0000,3E64,2A7C,0000,317A,3E3C,002D
229  DATA 101D,3F00,6100,2720,51CF,FFF6,2A7C,0000
230  DATA 341C,3E3C,000D,101D,3F00,6100,270A,51CF
231  DATA FFF6,6100,2630,6100,262C,2879,0000,3DF6
232  DATA 2A4C,33F9,0000,3E08,0000,3E06,363C,000F
233  DATA 3803,3A39,0000,3E14,3604,6100,24C4,6100
234  DATA 24E4,3604,284D,3E3C,0005,3F3C,0020,6100
235  DATA 26C6,51CF,FFF6,6100,24F2,DBFC,0000,0010
236  DATA 0679,0010,0000,3E06,6100,25DA,51CD,FFCA
237  DATA 6100,268C,4CF9,38F8,0000,3E64,33FC,0002
238  DATA 0000,3A1C,6000,FE9A,4E75,6100,25EE,207C
```

```
239  DATA 0000,322F,6100,21A4,33FC,0000,0000,3E02
240  DATA 33FC,0004,0000,3E04,6100,25F8,6100,25A8
241  DATA 3039,0000,3B48,B07C,0400,6612,33FC,0200
242  DATA 0000,3EA4,33FC,02D0,0000,3EA6,6010,33FC
243  DATA 0000,0000,3EA4,33FC,00D0,0000,3EA6,33FC
244  DATA 0012,0000,3E12,23FC,0000,AC1A,0000,3E60
245  DATA 6134,6100,F8AE,6100,F8AA,33FC,0002,0000
246  DATA 3E04,6100,259E,6100,2518,6100,256E,207C
247  DATA 0000,321B,6100,2124,6100,2554,6100,FD82
248  DATA 6100,25DC,4E75,48E7,1F1E,23F9,0000,3E60
249  DATA 0000,3DF6,33FC,0000,0000,3E08,33FC,0000
250  DATA 0000,3E06,6100,2318,6100,25B4,33FC,0007
251  DATA 0000,3E02,33FC,0004,0000,3E04,6100,2544
252  DATA 6100,2500,2F3C,0000,3E16,6100,25CE,6100
253  DATA 24FE,4A79,0000,3E16,6B4C,3039,0000,3E04
254  DATA 5940,E948,3439,0000,3E02,5F42,48C2,84FC
255  DATA 0003,D042,3239,0000,3E16,2679,0000,3DF6
256  DATA 1781,0000,6100,23D4,0C79,0034,0000,3E02
257  DATA 6D08,33FC,0004,0000,3E02,5679,0000,3E02
258  DATA 6100,24E0,609A,2039,0000,3EAC,4840,B03C
259  DATA 004B,673A,B03C,004D,675A,B03C,0050,6700
260  DATA 007A,B03C,0048,6700,011C,B03C,0052,6700
261  DATA 01BC,B03C,0072,6700,01B4,B03C,001C,6700
262  DATA 01AC,6100,2376,6100,249A,6000,FF54,3039
263  DATA 0000,3E02,B07C,0007,6E08,33FC,0037,0000
264  DATA 3E02,5779,0000,3E02,6100,2478,6100,24D0
265  DATA 6000,FF2E,3039,0000,3E02,B07C,0034,6D08
266  DATA 33FC,0004,0000,3E02,5679,0000,3E02,6100
267  DATA 2452,6100,24AA,6000,FF08,6100,2412,3039
268  DATA 0000,3E04,B07C,0016,6D00,0088,3039,0000
269  DATA 3E08,B079,0000,3EA4,6638,0679,00D0,0000
270  DATA 3E08,06B9,0000,00D0,0000,3DF6,33F9,0000
271  DATA 3E02,0000,3E10,6100,21C6,33F9,0000,3E10
272  DATA 0000,3E02,33FC,0005,0000,3E04,6100,23F4
273  DATA 604A,B079,0000,3EA6,6742,0679,0100,0000
274  DATA 3E08,06B9,0000,0100,0000,3DF6,33F9,0000
275  DATA 3E02,0000,3E10,6100,2186,33F9,0000,3E10
276  DATA 0000,3E02,33FC,0006,0000,3E04,6100,23B4
277  DATA 600A,5279,0000,3E04,6100,23A8,6100,2400
278  DATA 6000,FE5E,6100,2368,3039,0000,3E04,B07C
```

```
279  DATA 0004,6600,0086,3039,0000,3E08,B07C,0000
280  DATA 6700,0082,B079,0000,3EA6,6738,0479,0100
281  DATA 0000,3E08,04B9,0000,0100,0000,3DF6,33F9
282  DATA 0000,3E02,0000,3E10,6100,2114,33F9,0000
283  DATA 3E10,0000,3E02,33FC,0013,0000,3E04,6100
284  DATA 2342,6040,0479,00D0,0000,3E08,04B9,0000
285  DATA 00D0,0000,3DF6,33F9,0000,3E02,0000,3E10
286  DATA 6100,20DC,33F9,0000,3E10,0000,3E02,33FC
287  DATA 0013,0000,3E04,6100,230A,5379,0000,3E04
288  DATA 6100,2300,6100,2358,6000,FDB6,33FC,0000
289  DATA 0000,3E02,33FC,0004,0000,3E04,6100,22E4
290  DATA 4CDF,78F8,4E75,48E7,1F1E,33FC,0000,0000
291  DATA 3E02,33FC,0002,0000,3E04,6100,22C6,207C
292  DATA 0000,31CD,6100,1E54,267C,0000,317A,363C
293  DATA 002D,101B,3F00,6100,231E,51CB,FFF6,207C
294  DATA 0000,31E9,6100,1E34,6100,22F4,6100,2322
295  DATA B03C,0079,6706,B03C,0059,6660,3039,0000
296  DATA 3B48,B07C,0400,6700,0440,323C,0001,3F01
297  DATA 3F39,0000,3DE6,3F39,0000,3DE2,3F39,0000
298  DATA 3DE4,3F39,0000,3DE8,42A7,2F3C,0000,AC1A
299  DATA 3F3C,0009,4E4E,DFFC,0000,0014,4A40,6B34
300  DATA 6100,21BE,6100,2298,6100,2210,207C,0000
301  DATA 321B,6100,1DC6,4CDF,78F8,4E75,6100,21A2
302  DATA 207C,0000,31FB,6100,1DB2,6100,2272,6100
303  DATA 22A0,60CC,3F00,6100,F56E,60C4,3039,0000
304  DATA 3DE8,3F39,0000,3DE8,3F3C,0007,4E4D,588F
305  DATA 4A80,6608,3F00,6100,F54E,6044,2040,33D8
306  DATA 0000,3E34,33D8,0000,3E36,33D8,0000,3E38
307  DATA 33D8,0000,3E3A,33D8,0000,3E3C,33D8,0000
308  DATA 3E3E,33D8,0000,3E40,33D8,0000,3E42,33D8
309  DATA 0000,3E44,33D8,0000,3E4E,33D8,0000,3E4E
310  DATA 4E75,3F39,0000,3DE8,3F39,0000,3E3E,3F39
311  DATA 0000,3E3C,2F3C,0000,5DFA,3F3C,0002,3F3C
312  DATA 0004,4E4D,DFFC,0000,000E,4A40,6B02,4E75
313  DATA 3F00,6100,F4D2,60F6,3F39,0000,3DE8,3039
314  DATA 0000,3E3C,E348,5240,3F00,3F39,0000,3E3A
315  DATA 2F3C,0000,3EBA,3F3C,0002,3F3C,0004,4E4D
316  DATA DFFC,0000,000E,4A40,6B02,4E75,3F00,6100
317  DATA F496,60F6,3039,0000,3DEC,B07C,0063,6D06
318  DATA 303C,0000,6002,5240,33C0,0000,3DEC,48C0
```

```
319  DATA 80FC,000A,D03C,0030,13C0,0000,391A,4840
320  DATA D03C,0030,13C0,0000,391B,6100,1E50,4E75
321  DATA 3039,0000,3DEC,B07C,0000,6F04,5340,6004
322  DATA 303C,0063,33C0,0000,3DEC,48C0,80FC,000A
323  DATA D03C,0030,13C0,0000,391A,4840,D03C,0030
324  DATA 13C0,0000,391B,6100,1E14,4E75,3039,0000
325  DATA 3DEE,B07C,0063,6D06,303C,0000,6002,5240
326  DATA 33C0,0000,3DEE,48C0,80FC,000A,D03C,0030
327  DATA 13C0,0000,392B,4840,D03C,0030,13C0,0000
328  DATA 392C,6100,1DD8,4E75,3039,0000,3DEE,B07C
329  DATA 0000,6F04,5340,6004,303C,0063,33C0,0000
330  DATA 3DEE,48C0,80FC,000A,D03C,0030,13C0,0000
331  DATA 392B,4840,D03C,0030,13C0,0000,392C,6100
332  DATA 1D9C,4E75,6100,FE36,6100,FE98,6100,FECA
333  DATA 6106,6100,F32E,4E75,33FC,0004,0000,3E04
334  DATA 33FC,000A,0000,3E02,6100,2018,207C,0000
335  DATA 396D,6100,1BA6,33FC,002A,0000,3E50,33FC
336  DATA 0006,0000,3E04,33FC,000C,0000,3E02,6100
337  DATA 1FF2,207C,0000,358F,6100,1B80,6100,2010
338  DATA 3F39,0000,3E34,6100,1C12,5279,0000,3E04
339  DATA 33FC,000C,0000,3E02,6100,1FC8,207C,0000
340  DATA 35A4,6100,1B56,6100,1FE6,3F39,0000,3E36
341  DATA 6100,1BE8,5279,0000,3E04,33FC,000C,0000
342  DATA 3E02,6100,1F9E,207C,0000,35BB,6100,1B2C
343  DATA 6100,1FBC,3F39,0000,3E38,6100,1BBE,5279
344  DATA 0000,3E04,33FC,000C,0000,3E02,6100,1F74
345  DATA 207C,0000,35D1,6100,1B02,6100,1F92,3F39
346  DATA 0000,3E3A,6100,1B94,5279,0000,3E04,33FC
347  DATA 000C,0000,3E02,6100,1F4A,207C,0000,35EA
348  DATA 6100,1AD8,6100,1F68,3F39,0000,3E3C,6100
349  DATA 1B6A,5279,0000,3E04,33FC,000C,0000,3E02
350  DATA 6100,1F20,207C,0000,35FD,6100,1AAE,6100
351  DATA 1F3E,3F39,0000,3E3E,6100,1B40,5279,0000
352  DATA 3E04,33FC,000C,0000,3E02,6100,1EF6,207C
353  DATA 0000,3618,6100,1A84,6100,1F14,3F39,0000
354  DATA 3E40,6100,1B16,5279,0000,3E04,33FC,000C
355  DATA 0000,3E02,6100,1ECC,207C,0000,3637,6100
356  DATA 1A5A,6100,1EEA,3F39,0000,3E42,6100,1AEC
357  DATA 5279,0000,3E04,33FC,000C,0000,3E02,6100
358  DATA 1EA2,207C,0000,364E,6100,1A30,6100,1EC0
```

```
359  DATA 3F39,0000,3E4E,6100,1AC2,5479,0000,3E04
360  DATA 33FC,000A,0000,3E02,6100,1E78,207C,0000
361  DATA 3662,3039,0000,3E4E,B07C,0002,6606,207C
362  DATA 0000,36A1,6100,19F4,6100,1EB4,6100,1EE2
363  DATA 6100,1E28,6100,1DCA,6100,1E20,207C,0000
364  DATA 3954,6100,19D6,4E75,6100,04AA,50F9,0000
365  DATA 043E,6100,0544,6100,04E8,6100,0688,6150
366  DATA 51F9,0000,043E,6100,1E76,6100,1DEE,6100
367  DATA 1D90,6100,04CC,6100,04A0,207C,0000,3B0E
368  DATA 6100,1998,6100,1E8A,6100,046A,6100,0550
369  DATA 6100,0486,6100,1DC4,6100,1D66,6100,1DBC
370  DATA 207C,0000,3B33,6100,1972,4CDF,78F8,4E75
371  DATA 6100,06B0,33FC,0190,00FF,8606,33FC,0090
372  DATA 00FF,8606,33FC,0190,00FF,8606,3C3C,0004
373  DATA 6100,04A2,33FC,0184,00FF,8606,3C39,0000
374  DATA 3DE4,6100,0490,33FC,0180,00FF,8606,3C3C
375  DATA 00A0,6100,0480,2E3C,0005,0000,0839,0005
376  DATA 00FF,FA01,6716,5387,66F2,3F3C,FFF7,6100
377  DATA F0D6,6100,1D46,6100,1CE8,4E75,6100,0466
378  DATA 3039,0000,3A2A,0800,0006,6602,4E75,3F3C
379  DATA FFF8,6100,F0B2,6100,1D22,6100,1CC4,4E75
380  DATA 303C,0200,C0F9,0000,3E5A,33C0,0000,3E2E
381  DATA 3F39,0000,3E5A,3F39,0000,3DE6,3F39,0000
382  DATA 3DE2,3F3C,0001,3F39,0000,3DE8,42A7,2F3C
383  DATA 0000,AC1A,3F3C,0008,4E4E,DFFC,0000,0014
384  DATA 4A40,6B06,6100,0136,4E75,3F00,6100,F058
385  DATA 6100,1D4C,6100,1D7A,6100,1CC0,207C,0000
386  DATA 32E7,6100,1876,6100,1C4C,60DC,3039,0000
387  DATA 3E5A,B079,0000,3DEE,6D06,303C,0000,6002
388  DATA 5240,33C0,0000,3E5A,48C0,80FC,000A,D03C
389  DATA 0030,13C0,0000,32C1,4840,D03C,0030,13C0
390  DATA 0000,32C2,6100,19F6,4E75,3039,0000,3E5A
391  DATA B07C,0000,6F04,5340,6006,3039,0000,3DEE
392  DATA 33C0,0000,3E5A,48C0,80FC,000A,D03C,0030
393  DATA 13C0,0000,32C1,4840,D03C,0030,13C0,0000
394  DATA 32C2,6100,19B8,4E75,33FC,0000,0000,3EA4
395  DATA 33FC,00D0,0000,3EA6,33FC,0012,0000,3E12
396  DATA 2039,0000,3DF6,90BC,0000,AC1A,80FC,0200
397  DATA 4840,4A40,670A,04B9,0000,0100,0000,3DF6
398  DATA 23F9,0000,3DF6,0000,3E60,33FC,0000,0000
```

```
399  DATA 3E08,33FC,0014,0000,3E02,33FC,0002,0000
400  DATA 3E04,6100,1BFE,207C,0000,322F,6100,178C
401  DATA 6100,F674,6100,1BC4,6100,1B66,6100,1BBC
402  DATA 207C,0000,32E7,6100,1772,6100,EED6,6100
403  DATA EED2,6100,1BBA,6100,1B7E,4E75,33FC,0000
404  DATA 0000,3E08,23FC,0000,AC1A,0000,3DF6,33FC
405  DATA 0000,0000,3E4C,33FC,000F,0000,3E12,33FC
406  DATA 0002,0000,3E04,33FC,003B,0000,3E02,6100
407  DATA 1B92,207C,0000,3317,6100,1720,4240,303C
408  DATA 0001,3F00,6100,17B4,207C,0000,3324,6100
409  DATA 170A,33FC,0004,0000,3E04,33FC,0000,0000
410  DATA 3E02,6100,1B5E,6100,1B0E,6100,1BB2,6100
411  DATA 190E,6100,1B88,4840,B03C,0048,672E,B03C
412  DATA 0050,6700,0092,B03C,001C,6700,00FC,B03C
413  DATA 004B,6710,B03C,004D,6702,60D6,6100,EE46
414  DATA 6000,00E6,6100,EE1C,6000,00DE,3039,0000
415  DATA 3E08,B07C,0000,6712,0479,0100,0000,3E08
416  DATA 04B9,0000,0100,0000,3DF6,3039,0000,3E08
417  DATA E048,E248,5240,33C0,0000,3EB4,33FC,003B
418  DATA 0000,3E02,33FC,0002,0000,3E04,6100,1AD4
419  DATA 207C,0000,3317,6100,1662,3F39,0000,3EB4
420  DATA 6100,16F8,207C,0000,3324,6100,164E,6100
421  DATA 1A24,6000,FF5A,3039,0000,3E08,3239,0000
422  DATA 3E2E,927C,0100,B041,6712,0679,0100,0000
423  DATA 3E08,06B9,0000,0100,0000,3DF6,3039,0000
424  DATA 3E08,E048,E248,5240,33C0,0000,3EB4,33FC
425  DATA 003B,0000,3E02,33FC,0002,0000,3E04,6100
426  DATA 1A62,207C,0000,3317,6100,15F0,3F39,0000
427  DATA 3EB4,6100,1686,207C,0000,3324,6100,15DC
428  DATA 6100,19B2,6000,FEE8,6100,1A94,4E75,2F0C
429  DATA 33FC,0002,0000,3E04,6100,1A28,6100,19A2
430  DATA 207C,0000,3328,6100,15B2,343C,0021,287C
431  DATA 0000,317A,101C,3F00,6100,1A7C,51CA,FFF6
432  DATA 207C,0000,3342,6100,1592,6100,1A52,6100
433  DATA 1A80,B03C,0059,6706,B03C,0079,6634,3F39
434  DATA 0000,3E5A,3F39,0000,3DE6,3F39,0000,3DE2
435  DATA 3F3C,0001,3F39,0000,3DE8,42A7,2F3C,0000
436  DATA AC1A,3F3C,0009,4E4E,DFFC,0000,0014,4A40
437  DATA 6B1A,6100,1986,6100,1928,6100,197E,207C
438  DATA 0000,32E7,6100,1534,285F,4E75,3F00,6100
```

```
439    DATA ECF6,60E6,2F3C,0000,0001,3F3C,0020,4E41
440    DATA 5C8F,4A40,6610,42A7,3F3C,0020,4E41,5C8F
441    DATA 23C0,0000,3E20,4E75,2F3C,0000,0001,3F3C
442    DATA 0020,4E41,5C8F,4A40,670E,2F39,0000,3E20
443    DATA 3F3C,0020,4E41,5C8F,4E75,51CF,FFFE,4E75
444    DATA 61B2,33FC,0080,00FF,8606,3C3C,00D0,6124
445    DATA 3E3C,0028,61E4,4E75,619A,3639,00FF,8604
446    DATA 4E71,40E7,3F07,3E3C,0028,51CF,FFFE,3E1F
447    DATA 46DF,4E75,6100,FF7E,61E8,33C6,00FF,8604
448    DATA 61E0,4E75,6100,FF6E,61D8,33F9,00FF,8604
449    DATA 0000,3A2A,61CC,4E75,6100,FF5A,3039,0000
450    DATA 3DE8,B07C,0001,6E34,5200,E308,8079,0000
451    DATA 3DE6,0A00,0007,C03C,0007,40E7,007C,0700
452    DATA 13FC,000E,00FF,8800,1239,00FF,8800,C23C
453    DATA 00F8,8200,13C1,00FF,8802,46DF,4E75,6100
454    DATA FF14,33FC,0080,00FF,8606,103C,0007,61CA
455    DATA 4E75,6100,FF00,42B9,0000,3E2A,40F9,0000
456    DATA 3EB4,46FC,2700,33FC,0090,00FF,8606,33FC
457    DATA 0190,00FF,8606,33FC,0090,00FF,8606,3C3C
458    DATA 0016,343C,0200,C4C6,33C2,0000,3E2E,D4BC
459    DATA 0000,AC1A,23C2,0000,3E5C,6100,FF38,203C
460    DATA 0000,AC1A,13C0,00FF,860D,E088,13C0,00FF
461    DATA 860B,E088,13C0,00FF,8609,33FC,0080,00FF
462    DATA 8606,3C3C,00E8,6100,FF0C,2E3C,0005,0000
463    DATA 2A79,0000,3E5C,303C,0200,51C8,FFFE,0839
464    DATA 0005,00FF,FA01,672A,5387,6756,13F9,00FF
465    DATA 8609,0000,3E2B,13F9,00FF,860B,0000,3E2C
466    DATA 13F9,00FF,860D,0000,3E2D,BBF9,0000,3E2A
467    DATA 6ECC,33FC,0090,00FF,8606,3A39,00FF,8606
468    DATA 33C5,0000,3E28,0805,0000,6714,33FC,0080
469    DATA 00FF,8606,6100,FEAE,46F9,0000,3EB4,4E75
470    DATA 60F6,60F4,6100,FE0E,6142,33FC,0086,00FF
471    DATA 8606,3C39,0000,3DE2,6100,FE7A,33FC,0080
472    DATA 00FF,8606,3C3C,001B,6100,FE6A,2E3C,0006
473    DATA 0000,5387,670C,0839,0005,00FF,FA01,66F2
474    DATA 4E75,3F3C,FFF9,6100,EABE,4E75,3C39,0000
475    DATA 3A26,CC7C,0003,2E3C,0005,0000,33FC,0080
476    DATA 00FF,8606,6100,FE2E,5387,670C,0839,0005
477    DATA 00FF,FA01,66F2,4E75,3F3C,FFF9,6100,EA88
478    DATA 4E75,203C,0000,AC1A,13C0,00FF,860D,E088
```

```
479   DATA 13C0,00FF,860B,E088,13C0,00FF,8609,4E75
480   DATA 48E7,1F1E,6100,16D4,6100,1676,6100,16CC
481   DATA 207C,0000,32FA,6100,1282,33FC,0012,0000
482   DATA 3E12,6100,FD50,50F9,0000,043E,6100,FDEA
483   DATA 6100,FD8E,6100,FF2E,6100,FE38,6100,FE34
484   DATA 6100,FD7E,6100,FD52,6118,6100,FD28,6100
485   DATA FE0E,51F9,0000,043E,6100,FD3E,4CDF,78F8
486   DATA 4E75,33FC,0000,0000,3E08,23FC,0000,AC1A
487   DATA 0000,3DF6,33FC,0012,0000,3E12,33FC,0064
488   DATA 0000,3E14,33FC,1E00,0000,3EA4,33FC,1ED0
489   DATA 0000,3EA6,6100,1658,6100,161C,6100,EEBC
490   DATA 6100,16BC,4E75,6100,1632,6100,15D4,207C
491   DATA 0000,3BC0,6100,11E4,3039,0000,3DE8,B07C
492   DATA 0002,6E00,008A,6100,FCAC,6100,FD4C,6100
493   DATA FCF0,6100,FE90,6100,FE8C,6100,FF06,611A
494   DATA 6100,FCDE,6100,FCB2,6100,006E,6100,FC86
495   DATA 6100,FD6C,6100,FCA2,4E75,6100,FC78,33FC
496   DATA 0090,00FF,8606,33FC,0190,00FF,8606,33FC
497   DATA 0090,00FF,8606,3C3C,0001,6100,FCD8,33FC
498   DATA 0080,00FF,8606,383C,0018,3C3C,00C8,2E3C
499   DATA 0004,0000,6100,FCBE,0839,0005,00FF,FA01
500   DATA 6706,5387,6708,60F0,51CC,FFE0,4E75,3F3C
501   DATA FFFA,6100,E912,4E75,6100,1580,6100,1522
502   DATA 207C,0000,3B54,6100,1132,6100,1582,3A3C
503   DATA 0011,267C,0000,AC1A,383C,0002,3F3C,0020
504   DATA 6100,15F4,101B,3F00,6100,11B0,3F3C,0020
505   DATA 6100,15E4,3F3C,0020,6100,15DC,51CC,FFE6
506   DATA 3F3C,0020,6100,15D0,3F3C,0020,6100,15C8
507   DATA 101B,4880,323C,0080,B07C,0000,6718,323C
508   DATA 0100,B07C,0001,670E,323C,0200,B07C,0002
509   DATA 6704,323C,0400,3F01,6100,1160,3F3C,0020
510   DATA 6100,1594,207C,0000,3BB4,6100,10AE,101B
511   DATA 3F00,6100,10D8,101B,3F00,6100,10D0,3F3C
512   DATA 000D,6100,1572,3F3C,000A,6100,156A,51CD
513   DATA FF68,6100,157C,4E75,6100,14C0,6100,1462
514   DATA 33FC,0014,0000,3E02,33FC,0002,0000,3E04
515   DATA 6100,14D0,207C,0000,322F,6100,105E,33FC
516   DATA 0200,0000,3EA4,33FC,02D0,0000,3EA6,23FC
517   DATA 0000,AC1A,0000,3E60,6100,EF2C,6100,147C
518   DATA 6100,141E,6100,1474,207C,0000,3449,6100
```

```
519   DATA 102A,6100,E78E,6100,E78A,6100,E786,6100
520   DATA 03AA,4E75,3F3C,FFFF,3F3C,000B,4E4D,588F
521   DATA 0800,0000,660C,0800,0001,6606,343C,0001
522   DATA 6004,343C,000A,3039,0000,3DEA,9042,B07C
523   DATA 0000,6D02,6006,3039,0000,3DF4,33C0,0000
524   DATA 3DEA,48C0,80FC,03E8,D03C,0030,13C0,0000
525   DATA 3424,4840,48C0,80FC,0064,D03C,0030,13C0
526   DATA 0000,3425,4840,48C0,80FC,000A,D03C,0030
527   DATA 13C0,0000,3426,4840,D03C,0030,13C0,0000
528   DATA 3427,6100,1158,4E75,3F3C,FFFF,3F3C,000B
529   DATA 4E4D,588F,0800,0000,660C,0800,0001,6606
530   DATA 343C,0001,6004,343C,000A,3039,0000,3DEA
531   DATA D042,B079,0000,3DF4,6D04,303C,0000,33C0
532   DATA 0000,3DEA,48C0,80FC,03E8,D03C,0030,13C0
533   DATA 0000,3424,4840,48C0,80FC,0064,D03C,0030
534   DATA 13C0,0000,3425,4840,48C0,80FC,000A,D03C
535   DATA 0030,13C0,0000,3426,4840,D03C,0030,13C0
536   DATA 0000,3427,6100,10D6,4E75,3039,0000,3DEA
537   DATA 33C0,0000,3E52,6148,3039,0000,3E54,4A40
538   DATA 6722,B07C,0FF8,6C1C,5340,33C0,0000,3DEA
539   DATA 23FC,0000,0003,0000,3DDA,6100,FF4C,6100
540   DATA 015A,4E75,6100,1314,3F3C,FFED,6100,E698
541   DATA 6100,1308,207C,0000,3449,6100,0EBE,60E2
542   DATA 207C,0000,5DFA,3039,0000,3E52,323C,0003
543   DATA C2C0,E249,0800,0000,6616,1030,1001,E148
544   DATA 8030,1000,C07C,0FFF,33C0,0000,3E54,6016
545   DATA 1030,1001,E148,1030,1000,E848,C07C,0FFF
546   DATA 33C0,0000,3E54,4E75,48E7,1C1C,33FC,0000
547   DATA 0000,3E02,33FC,0002,0000,3E04,6100,12C4
548   DATA 207C,0000,3572,6100,0E52,267C,0000,317A
549   DATA 363C,0009,101B,3F00,6100,131C,51CB,FFF6
550   DATA 267C,0000,341C,363C,000C,101B,3F00,6100
551   DATA 1306,51CB,FFF6,207C,0000,31E9,6100,0E1C
552   DATA 6100,12DC,6100,130A,B03C,0079,6706,B03C
553   DATA 0059,6660,3F39,0000,3DE8,3039,0000,3DEA
554   DATA 5540,C1F9,0000,3E36,D079,0000,3E40,3F00
555   DATA 3F39,0000,3E36,2F3C,0000,AC1A,3F3C,0003
556   DATA 3F3C,0004,4E4D,DFFC,0000,000E,4A40,6B1C
557   DATA 6100,1208,6100,11AA,6100,1200,207C,0000
558   DATA 3449,6100,0DB6,4CDF,3838,4E75,3F00,6100
```

```
559   DATA E576,60DC,6100,11E4,6100,1186,6100,11DC
560   DATA 207C,0000,31FB,6100,0D92,6100,1252,6100
561   DATA 1280,6100,116C,60B8,4E75,48E7,1F1E,3F39
562   DATA 0000,3DE8,3039,0000,3DEA,5540,C1F9,0000
563   DATA 3E36,D079,0000,3E40,4A40,6A04,303C,0000
564   DATA 33C0,0000,3E58,3F00,3F3C,0002,2F3C,0000
565   DATA AC1A,3F3C,0000,3F3C,0004,4E4D,DFFC,0000
566   DATA 000E,4A40,6B5C,3039,0000,3E58,81FC,0009
567   DATA 4840,5240,33C0,0000,3DE4,4840,3400,33FC
568   DATA 0000,0000,3DE6,3239,0000,3E4E,B27C,0002
569   DATA 6610,E248,0802,0000,6708,33FC,0001,0000
570   DATA 3DE6,33C0,0000,3DE2,612A,6100,007E,6100
571   DATA 112A,207C,0000,3449,6100,0CE0,4CDF,78F8
572   DATA 4E75,6100,EF38,4A80,6600,FF54,3F00,6100
573   DATA E496,60DA,3039,0000,3DE6,D03C,0030,13C0
574   DATA 0000,318C,3039,0000,3DE4,48C0,81FC,000A
575   DATA D03C,0030,13C0,0000,31A4,4840,D03C,0030
576   DATA 13C0,0000,31A5,3039,0000,3DE2,48C0,81FC
577   DATA 000A,D03C,0030,13C0,0000,3197,4840,D03C
578   DATA 0030,13C0,0000,3198,4E75,33FC,0000,0000
579   DATA 3E08,33FC,0012,0000,3E12,33FC,003F,0000
580   DATA 3E14,23FC,0000,AC1A,0000,3DF6,33FC,0200
581   DATA 0000,3EA4,33FC,02D0,0000,3EA6,6100,1090
582   DATA 6100,1054,33FC,0000,0000,3E02,33FC,0018
583   DATA 0000,3E04,6100,108C,207C,0000,345E,6100
584   DATA 0C1A,6100,E8D6,4E75,48E7,1F1C,6100,EE6E
585   DATA 6100,EED0,6100,EF02,33FC,0000,0000,3E02
586   DATA 33FC,0002,0000,3E04,6100,1058,207C,0000
587   DATA 34CF,6100,0BE6,33FC,0011,0000,3E12,6100
588   DATA 0FB4,267C,0000,3EBA,284B,23CB,0000,3DF6
589   DATA 23CB,0000,3DFA,6100,1016,6100,027E,6100
590   DATA 100E,23FC,0000,3EBA,0000,3DF6,6100,0F6E
591   DATA 6100,0334,6100,0F72,6100,1042,4840,B03C
592   DATA 001C,6700,0182,B03C,0048,6724,B03C,0050
593   DATA 6700,00AA,B03C,004B,670E,B03C,004D,66D8
594   DATA 6100,E302,6000,0144,6100,E2D8,6000,013C
595   DATA 3039,0000,3E04,B07C,0004,6F30,33FC,0000
596   DATA 0000,3E02,6100,0FBC,6100,02DC,5379,0000
597   DATA 3E04,33FC,0000,0000,3E02,6100,0FA6,6100
598   DATA 0EFC,6100,02C2,6100,0F00,604C,0CB9,0000
```

```
599   DATA 3EBA,0000,3DF6,6740,2039,0000,3DF6,3039
600   DATA 0000,3E12,5240,C1FC,0020,91B9,0000,3DF6
601   DATA 6100,01C8,33FC,0015,0000,3E04,33FC,0000
602   DATA 0000,3E02,6100,0F5C,6100,0EB2,6100,0278
603   DATA 6100,0EB6,6100,0FA8,6000,FF3E,3039,0000
604   DATA 3E04,B07C,0014,6E50,3039,0000,3E04,5240
605   DATA 5940,48C0,EB88,2C79,0000,3DF6,1036,0800
606   DATA 6700,0084,33FC,0000,0000,3E02,6100,0F14
607   DATA 6100,0234,5279,0000,3E04,33FC,0000,0000
608   DATA 3E02,6100,0EFE,6100,0E54,6100,021A,6100
609   DATA 0E58,6100,0EEE,604E,3039,0000,3E04,5240
610   DATA 5940,48C0,EB88,2C79,0000,3DF6,1036,0800
611   DATA 6734,3039,0000,3E12,5240,C1FC,0020,D0B9
612   DATA 0000,3DF6,23C0,0000,3DF6,6100,0EA2,6100
613   DATA 010A,6100,0E9A,6100,0E04,6100,01CA,6100
614   DATA 0E08,6100,0EFA,6000,FE90,6100,0E6E,6100
615   DATA 0E10,6100,0E66,207C,0000,3449,6100,0A1C
616   DATA 4CDF,38F8,4E75,2079,0000,3DF6,3039,0000
617   DATA 3E04,5940,48C0,EB88,1230,080B,B23C,0010
618   DATA 6726,3039,0000,3E56,5340,33C0,0000,3DEA
619   DATA 23FC,0000,0003,0000,3DDA,6100,FA4C,60AA
620   DATA 6100,ECD6,6000,0086,4A79,0000,3E56,67F0
621   DATA 3039,0000,3E56,23FC,0000,3EBA,0000,3EB6
622   DATA 4243,3039,0000,3E56,3F39,0000,3DE8,5540
623   DATA C1F9,0000,3E36,D079,0000,3E40,3F00,3F3C
624   DATA 0002,2F39,0000,3EB6,3F3C,0002,3F3C,0004
625   DATA 4E4D,DFFC,0000,000E,4A40,6B34,06B9,0000
626   DATA 0400,0000,3EB6,33F9,0000,3E56,0000,3E52
627   DATA 6100,FAAE,3039,0000,3E54,33C0,0000,3E56
628   DATA 4A40,6708,B07C,0FF8,6C02,6096,6000,FD4A
629   DATA 3F00,6100,E112,6000,FF12,33FC,0000,0000
630   DATA 3E4A,6100,0D8A,6100,0D4E,2A79,0000,3DF6
631   DATA 3E39,0000,3E12,103C,0020,3F00,6100,0DF8
632   DATA 103C,0020,3F00,6100,0DEE,4284,3C3C,0009
633   DATA 1035,4800,6700,0086,5284,3F00,6100,0DD8
634   DATA 1035,4800,5284,3F00,6100,0DCC,51CE,FFF2
635   DATA 33FC,0014,0000,3E50,6100,0D74,6100,015E
636   DATA 33FC,0028,0000,3E50,6100,0D64,6100,00DC
637   DATA 33FC,0037,0000,3E50,6100,0D54,6100,0104
638   DATA 33FC,0000,0000,3E02,5279,0000,3E04,6100
```

```
639   DATA 0D12,DBFC,0000,0020,51CF,FF7C,33FC,0000
640   DATA 0000,3E02,33FC,0018,0000,3E04,6100,0CF4
641   DATA 207C,0000,3520,6100,0882,4E75,33FC,0001
642   DATA 0000,3E4A,60D6,2679,0000,3DF6,33FC,0000
643   DATA 0000,3E4A,3639,0000,3E04,5943,48C3,EB8B
644   DATA 183C,0020,3F04,6100,0D2E,3F04,6100,0D28
645   DATA 1033,3800,674A,3F00,6100,0D1C,5283,3C3C
646   DATA 0009,1033,3800,5283,3F00,6100,0D0A,51CE
647   DATA FFF2,33FC,0014,0000,3E50,6100,0CB2,6100
648   DATA 009C,33FC,0028,0000,3E50,6100,0CA2,611A
649   DATA 33FC,0037,0000,3E50,6100,0C94,6144,4E75
650   DATA 33FC,0001,0000,3E4A,60F4,103C,0020,3F00
651   DATA 6100,0CC4,3039,0000,3E04,5940,48C0,EB88
652   DATA 1233,081B,E149,1233,081A,33C1,0000,3E56
653   DATA 3F01,6100,0866,103C,0020,3F00,6100,0C98
654   DATA 4E75,3639,0000,3E04,5943,48C3,EB8B,4281
655   DATA 2679,0000,3DF6,1233,381F,E189,1233,381E
656   DATA E189,1233,381D,E189,1233,381C,2F01,6100
657   DATA 08B2,3F3C,0020,6100,0C5E,4E75,3639,0000
658   DATA 3E04,5943,48C3,EB8B,2679,0000,3DF6,1233
659   DATA 3800,B23C,00E5,6758,1233,380B,B23C,0010
660   DATA 672A,B23C,0001,6730,B23C,0002,6736,B23C
661   DATA 0008,670C,207C,0000,36F1,6100,073E,4E75
662   DATA 207C,0000,3735,6100,0732,60F2,207C,0000
663   DATA 36E0,6100,0726,60E6,207C,0000,3702,6100
664   DATA 071A,60DA,207C,0000,3713,6100,070E,60CE
665   DATA 207C,0000,3724,6100,0702,60C2,6100,0B3C
666   DATA 6100,0ADE,6100,0BB8,207C,0000,37E4,6100
667   DATA 06EA,3F39,0000,3DE2,6100,0780,207C,0000
668   DATA 382E,6100,06D6,3F39,0000,3DE6,6100,076C
669   DATA 207C,0000,3839,6100,06C2,3F39,0000,3DE8
670   DATA 6100,0758,207C,0000,37EF,6100,06AE,6100
671   DATA 0BA0,B03C,0079,671E,B03C,0059,6718,6100
672   DATA 0A80,6100,0B5A,207C,0000,380C,6100,068C
673   DATA 6100,0B7E,603E,3F3C,E5E5,2F3C,8765,4321
674   DATA 3F3C,0001,3F39,0000,3DE6,3F39,0000,3DE2
675   DATA 3F39,0000,3E5A,3F39,0000,3DE8,42A7,2F3C
676   DATA 0000,7D3A,3F3C,000A,4E4E,DFFC,0000,001A
677   DATA 4A40,6B1C,6100,0A84,6100,0A26,6100,0A7C
678   DATA 207C,0000,37CA,6100,0632,6100,DDB8,4E75
```

```
679   DATA 3F00,6100,DDF2,60DC,33FC,0001,0000,3A24
680   DATA 247C,0000,7D3A,3039,0000,3B4A,3E3C,004E
681   DATA 6100,00D4,3039,0000,3B4C,3E3C,0000,6100
682   DATA 00C6,303C,0003,1E3C,00F5,6100,00BA,14FC
683   DATA 00FE,3039,0000,3DE2,14C0,3039,0000,3DE6
684   DATA 14C0,3039,0000,3A24,14C0,3039,0000,3B48
685   DATA B07C,0400,671E,B07C,0200,6712,B07C,0100
686   DATA 6706,323C,0000,6010,323C,0001,600A,323C
687   DATA 0002,6004,323C,0003,14C1,14FC,00F7,3039
688   DATA 0000,3B4E,3E3C,004E,615C,3039,0000,3B4C
689   DATA 3E3C,0000,6150,303C,0003,3E3C,00F5,6146
690   DATA 14FC,00FB,3039,0000,3B48,1E3C,00E5,6136
691   DATA 14FC,00F7,3039,0000,3B50,3E3C,004E,6126
692   DATA 3039,0000,3A24,5240,33C0,0000,3A24,B079
693   DATA 0000,3E5A,6F00,FF3E,3039,0000,3B52,3E3C
694   DATA 004E,6102,4E75,5340,14C7,51C8,FFFC,4E75
695   DATA 203C,0000,7D3A,13C0,00FF,860D,E088,13C0
696   DATA 00FF,860B,E088,13C0,00FF,8609,4E75,33FC
697   DATA 0190,00FF,8606,33FC,0090,00FF,8606,33FC
698   DATA 0190,00FF,8606,3C3C,001F,6100,F048,33FC
699   DATA 0180,00FF,8606,3C3C,00F8,6100,F038,2E3C
700   DATA 0006,0000,5387,670C,0839,0005,00FF,FA01
701   DATA 66F2,4E75,3F3C,FFE8,6100,DC8C,4E75,48E7
702   DATA 1F1E,6100,08F6,6100,0898,207C,0000,3845
703   DATA 6100,04A8,6100,0968,6100,0996,B03C,0079
704   DATA 6708,B03C,0059,6600,0064,6100,EF68,50F9
705   DATA 0000,043E,6100,F1CC,6100,EFFE,6100,EFA2
706   DATA 6100,F142,6100,F04C,6100,FF46,6100,FE4A
707   DATA 6100,F132,6100,FF58,6100,0924,6100,089C
708   DATA 6100,083E,6100,EF7A,6100,EF4E,207C,0000
709   DATA 388D,6100,0446,6100,0938,6100,EF18,51F9
710   DATA 0000,043E,6100,EFF8,6100,EF2E,6100,086C
711   DATA 6100,080E,6100,0864,207C,0000,3AD0,6100
712   DATA 041A,4CDF,78F8,4E75,B07C,0063,6D06,303C
713   DATA 0000,6002,5240,4E75,3039,0000,3B4A,61E8
714   DATA 33C0,0000,3B4A,80FC,000A,D03C,0030,13C0
715   DATA 0000,3A87,4840,D03C,0030,13C0,0000,3A88
716   DATA 6100,059A,4E75,3039,0000,3B4C,61BA,33C0
717   DATA 0000,3B4C,80FC,000A,D03C,0030,13C0,0000
718   DATA 3A92,4840,D03C,0030,13C0,0000,3A93,6100
```

```
719   DATA 056C,4E75,3039,0000,3B4E,618C,33C0,0000
720   DATA 3B4E,80FC,000A,D03C,0030,13C0,0000,3A9D
721   DATA 4840,D03C,0030,13C0,0000,3A9E,6100,053E
722   DATA 4E75,3039,0000,3B50,6100,FF5E,33C0,0000
723   DATA 3B50,80FC,000A,D03C,0030,13C0,0000,3AA8
724   DATA 4840,D03C,0030,13C0,0000,3AA9,6100,050E
725   DATA 4E75,3F3C,FFFF,3F3C,000B,4E4D,588F,323C
726   DATA 000A,0800,0000,660A,0800,0001,6604,323C
727   DATA 0001,3039,0000,3B52,D041,B07C,03E7,6F04
728   DATA 303C,0000,33C0,0000,3B52,48C0,81FC,0064
729   DATA D03C,0030,13C0,0000,3AB3,4840,48C0,81FC
730   DATA 000A,D03C,0030,13C0,0000,3AB4,4840,D03C
731   DATA 0030,13C0,0000,3AB5,6100,04A2,4E75,B07C
732   DATA 0000,6F04,5340,6004,303C,0063,4E75,3039
733   DATA 0000,3B4A,61E8,33C0,0000,3B4A,80FC,000A
734   DATA D03C,0030,13C0,0000,3A87,4840,D03C,0030
735   DATA 13C0,0000,3A88,6100,0464,4E75,3039,0000
736   DATA 3B4C,61BA,33C0,0000,3B4C,80FC,000A,D03C
737   DATA 0030,13C0,0000,3A92,4840,D03C,0030,13C0
738   DATA 0000,3A93,6100,0436,4E75,3039,0000,3B4E
739   DATA 618C,33C0,0000,3B4E,80FC,000A,D03C,0030
740   DATA 13C0,0000,3A9D,4840,D03C,0030,13C0,0000
741   DATA 3A9E,6100,0408,4E75,3039,0000,3B50,6100
742   DATA FF5E,33C0,0000,3B50,80FC,000A,D03C,0030
743   DATA 13C0,0000,3AA8,4840,D03C,0030,13C0,0000
744   DATA 3AA9,6100,03D8,4E75,3F3C,FFFF,3F3C,000B
745   DATA 4E4D,588F,323C,000A,0800,0000,660A,0800
746   DATA 0001,6604,323C,0001,3039,0000,3B52,9041
747   DATA 6A04,303C,03E7,33C0,0000,3B52,48C0,81FC
748   DATA 0064,D03C,0030,13C0,0000,3AB3,4840,48C0
749   DATA 81FC,000A,D03C,0030,13C0,0000,3AB4,4840
750   DATA D03C,0030,13C0,0000,3AB5,6100,0370,4E75
751   DATA 3039,0000,3B48,B07C,0080,6732,B07C,0100
752   DATA 6752,B07C,0200,6772,303C,0080,13FC,0030
753   DATA 0000,3AC3,13FC,0031,0000,3AC4,13FC,0032
754   DATA 0000,3AC5,13FC,0038,0000,3AC6,6070,303C
755   DATA 0100,13FC,0030,0000,3AC3,13FC,0032,0000
756   DATA 3AC4,13FC,0035,0000,3AC5,13FC,0036,0000
757   DATA 3AC6,604A,303C,0200,13FC,0030,0000,3AC3
758   DATA 13FC,0035,0000,3AC4,13FC,0031,0000,3AC5
```

```
759   DATA 13FC,0032,0000,3AC6,6024,303C,0400,13FC
760   DATA 0031,0000,3AC3,13FC,0030,0000,3AC4,13FC
761   DATA 0032,0000,3AC5,13FC,0034,0000,3AC6,33C0
762   DATA 0000,3B48,6100,02B6,4E75,3039,0000,3B48
763   DATA B07C,0080,6732,B07C,0100,6752,B07C,0200
764   DATA 6772,303C,0200,13FC,0030,0000,3AC3,13FC
765   DATA 0035,0000,3AC4,13FC,0031,0000,3AC5,13FC
766   DATA 0032,0000,3AC6,6070,303C,0400,13FC,0031
767   DATA 0000,3AC3,13FC,0030,0000,3AC4,13FC,0032
768   DATA 0000,3AC5,13FC,0034,0000,3AC6,604A,303C
769   DATA 0080,13FC,0030,0000,3AC3,13FC,0031,0000
770   DATA 3AC4,13FC,0032,0000,3AC5,13FC,0038,0000
771   DATA 3AC6,6024,303C,0100,13FC,0030,0000,3AC3
772   DATA 13FC,0032,0000,3AC4,13FC,0035,0000,3AC5
773   DATA 13FC,0036,0000,3AC6,33C0,0000,3B48,6100
774   DATA 01FC,4E75,2079,0000,3EA8,317C,0000,0026
775   DATA 317C,0014,0028,317C,027F,002A,317C,0001
776   DATA 0018,317C,0000,0024,217C,0000,3DD0,002E
777   DATA 317C,0000,0032,A004,4E75,2F08,3F3C,0009
778   DATA 4E41,5C8F,4E75,A000,23C8,0000,3EA8,317C
779   DATA 0000,0020,317C,FFFF,0022,317C,0000,0024
780   DATA 317C,0001,0018,4E75,61DC,4E75,322F,0004
781   DATA C27C,00FF,33C1,0000,3EB0,E849,4881,C27C
782   DATA 00FF,B27C,0009,6E04,6120,6002,6132,3239
783   DATA 0000,3EB0,C27C,000F,B27C,0009,6E04,610A
784   DATA 6002,611C,205F,548F,4ED0,D27C,0030,3F01
785   DATA 3F39,0000,3A1C,3F3C,0003,4E4D,5C8F,4E75
786   DATA 927C,000A,D27C,0041,3F01,3F39,0000,3A1C
787   DATA 3F3C,0003,4E4D,5C8F,4E75,33FC,0000,0000
788   DATA 3E48,362F,0004,48C3,87FC,2710,6708,33FC
789   DATA FFFF,0000,3E48,614E,4843,48C3,87FC,03E8
790   DATA 6708,33FC,FFFF,0000,3E48,613A,4843,48C3
791   DATA 87FC,0064,6708,33FC,FFFF,0000,3E48,6126
792   DATA 4843,48C3,87FC,000A,6708,33FC,FFFF,0000
793   DATA 3E48,6112,4843,33FC,FFFF,0000,3E48,6106
794   DATA 205F,548F,4ED0,4A79,0000,3E48,6608,103C
795   DATA 0020,3F00,6006,D63C,0030,3F03,6100,03B8
796   DATA 4E75,33FC,0000,0000,3E48,262F,0004,2803
797   DATA 87FC,2710,48C3,87FC,000A,3A03,4A43,6708
798   DATA 33FC,FFFF,0000,3E48,61BC,3605,C7FC,000A
```

```
799    DATA C7FC,2710,9883,2604,87FC,2710,6708,33FC
800    DATA FFFF,0000,3E48,619E,4843,48C3,87FC,03E8
801    DATA 6708,33FC,FFFF,0000,3E48,618A,4843,48C3
802    DATA 87FC,0064,6708,33FC,FFFF,0000,3E48,6100
803    DATA FF76,4843,48C3,87FC,000A,6708,33FC,FFFF
804    DATA 0000,3E48,6100,FF60,4843,33FC,FFFF,0000
805    DATA 3E48,6100,FF52,205F,588F,4ED0,33FC,0000
806    DATA 0000,3E02,33FC,0000,0000,3E04,6100,0294
807    DATA 2C79,0000,3DD2,2C39,0000,3DDA,5386,670C
808    DATA 5386,205E,6100,FE14,51CE,FFF8,6100,01CE
809    DATA 205E,6100,FE06,6100,01D0,2E39,0000,3DD6
810    DATA 9EB9,0000,3DDA,670C,5387,205E,6100,FDEC
811    DATA 51CF,FFF8,6100,FDAE,6100,01BA,4E75,48F9
812    DATA 38F8,0000,3E64,2879,0000,3DF6,2A4C,33F9
813    DATA 0000,3E08,0000,3E06,363C,000F,3803,3A39
814    DATA 0000,3E12,33FC,0004,0000,3E04,33FC,0004
815    DATA 0000,3E0A,33FC,0000,0000,3E02,6100,0204
816    DATA 33FC,0000,0000,3E02,33F9,0000,3E0A,0000
817    DATA 3E04,3604,6100,01EC,6146,6168,3604,284D
818    DATA 33FC,003B,0000,3E02,33F9,0000,3E0A,0000
819    DATA 3E04,6100,01CE,6100,0072,DBFC,0000,0010
820    DATA 5279,0000,3E0A,0679,0010,0000,3E06,51CD
821    DATA FFB0,6100,020A,4CF9,38F8,0000,3E64,4E75
822    DATA 3C39,0000,3E06,E04E,3F06,6100,FD60,3F39
823    DATA 0000,3E06,6100,FD56,1C3C,003A,3F06,6100
824    DATA 01F6,4E75,3F3C,0020,6100,01EC,3F3C,0020
825    DATA 6100,01E4,1E1C,3F07,6100,FD32,3F3C,0020
826    DATA 6100,01D4,51CB,FFEE,4E75,1E3C,003A,3F07
827    DATA 6100,01C4,3F3C,0020,6100,01BC,1E1C,BE3C
828    DATA 0020,6E04,1E3C,002E,4887,CE7C,00FF,3F07
829    DATA 6100,01A4,51CB,FFE6,4E75,33F9,0000,3E02
830    DATA 0000,3E10,33FC,0000,0000,3E02,6100,0114
831    DATA 33F9,0000,3E10,0000,3E02,363C,000F,3803
832    DATA 2879,0000,3DF6,4280,3039,0000,3E04,5940
833    DATA E948,3200,D079,0000,3E08,33C0,0000,3E06
834    DATA 48C1,D9C1,2A4C,6100,FF38,6100,FF58,33F9
835    DATA 0000,3E02,0000,3E10,33FC,003B,0000,3E02
836    DATA 6100,00C0,3604,284D,6100,FF60,33F9,0000
837    DATA 3E10,0000,3E02,6100,00AA,4E75,207C,0000
838    DATA 3B8E,6100,FC36,4E75,207C,0000,3B91,6100
```

```
839   DATA FC2A,4E75,207C,0000,3B8B,6100,FC1E,4E75
840   DATA 207C,0000,3BA8,6100,FC12,4E75,207C,0000
841   DATA 3B9C,6100,FC06,4E75,207C,0000,3B99,6100
842   DATA FBFA,4E75,3F3C,001C,6100,00CC,3F3C,000A
843   DATA 6100,00C4,4E75,207C,0000,3B88,6100,FBDC
844   DATA 4E75,207C,0000,3BB1,6100,FBD0,4E75,207C
845   DATA 0000,3BAE,6100,FBC4,4E75,33FC,001E,0000
846   DATA 3E02,33FC,0002,0000,3E04,6116,4E75,33FC
847   DATA 0000,0000,3E02,33FC,0004,0000,3E04,6102
848   DATA 4E75,207C,0000,3B94,5488,3039,0000,3E04
849   DATA D07C,0020,10C0,3039,0000,3E02,D07C,0020
850   DATA 10C0,207C,0000,3B94,6100,FB70,4E75,33F9
851   DATA 0000,3E50,0000,3E02,61C8,4E75,3F3C,0002
852   DATA 3F3C,0001,4E4D,588F,4A40,6A0E,3F3C,0002
853   DATA 3F3C,0002,4E4D,588F,4E75,7000,4E75,3F3C
854   DATA 000B,4E41,548F,4A40,670A,3F3C,0007,4E41
855   DATA 548F,60EA,4E75,302F,0004,3F00,3F39,0000
856   DATA 3A1C,3F3C,0003,4E4D,5C8F,205F,548F,4ED0
857   DATA 3F3C,0001,4E41,548F,4E75,61F4,23C0,0000
858   DATA 3EAC,B03C,0066,6E00,0078,B03C,0061,6D0A
859   DATA 903C,0061,D03C,000A,6010,B03C,0030,6D60
860   DATA B03C,0039,6E60,903C,0030,E948,33C0,0000
861   DATA 3EB0,61BC,23C0,0000,3EAC,B03C,0066,6E40
862   DATA B03C,0061,6D0A,903C,0061,D03C,000A,6010
863   DATA B03C,0030,6D2A,B03C,0039,6E40,903C,0030
864   DATA 3239,0000,3EB0,8041,4880,C07C,00FF,33C0
865   DATA 0000,3EB2,206F,0004,3080,205F,588F,4ED0
866   DATA 303C,FFFF,60EE,B03C,0046,6EF4,B03C,0041
867   DATA 6DEE,903C,0041,D03C,000A,608E,B03C,0046
868   DATA 6EDE,B03C,0041,6DD8,903C,0041,D03C,000A
869   DATA 60AE,0000,0290,0000,02DA,0000,0320,0000
870   DATA 0374,0000,03C2,0000,0456,0000,0186,0000
871   DATA 302E,0000,3038,0000,3046,0000,3050,0000
872   DATA 305B,0000,306E,0000,307A,2020,5452,4143
873   DATA 4B20,2000,2054,5241,434B,2F53,594E,4353
874   DATA 2000,2053,4543,544F,5220,2000,2043,4C55
875   DATA 5354,4552,2020,0020,464F,524D,4154,2020
876   DATA 0020,2046,4154,5320,2000,2020,4F50,5449
877   DATA 4F4E,5320,2000,2020,454E,4445,2020,001B
878   DATA 7020,2041,204C,4954,544C,4520,4449,534B
```

```
879   DATA 2055,5449,4C49,5459,2020,2028,4329,2055
880   DATA 2E20,4272,6175,6E20,3139,3836,2020,1B71
881   DATA 001B,7020,2020,2020,4441,5441,2042,4543
882   DATA 4B45,5220,464C,4F50,5059,2D42,5543,4820
883   DATA 469A,5220,4154,4152,4920,5354,2020,2020
884   DATA 1B71,001B,7020,2020,2020,5365,6C65,6374
885   DATA 204D,656E,7565,2049,7465,6D73,2077,6974
886   DATA 6820,4375,7273,6F72,2D4B,6579,7320,2020
887   DATA 2020,1B71,0000,0000,049C,0000,04F4,0000
888   DATA 054C,0000,05C8,0000,0644,0000,0BCA,0000
889   DATA 088E,0000,0254,0000,04C8,0000,0520,0000
890   DATA 058A,0000,0606,0000,0644,0000,0BCA,0000
891   DATA 088E,0000,0254,0000,317A,0000,3185,0000
892   DATA 318F,0000,319B,0000,31A8,0000,31B1,0000
893   DATA 31BB,0000,31C4,2064,7269,7665,3A20,3020
894   DATA 0020,7369,6465,3A20,3020,0020,7472,6163
895   DATA 6B3A,2030,3020,0020,7365,6374,6F72,3A20
896   DATA 3031,2000,2020,5245,4144,2020,0020,2057
897   DATA 5249,5445,2020,0020,2045,4449,5420,2000
898   DATA 2020,4241,434B,2020,001B,7020,5772,6974
899   DATA 6520,7468,6973,2053,6563,746F,7220,746F
900   DATA 3A20,1B71,001B,7020,203C,7965,732C,6E6F
901   DATA 3E20,3F20,1B71,001B,7020,4E6F,7420,7772
902   DATA 6974,7465,6E2E,2020,3C70,7265,7373,206B
903   DATA 6579,3E20,1B71,001B,7020,2053,4543,544F
904   DATA 5220,4D4F,4445,2020,1B71,001B,7020,2045
905   DATA 4449,5420,4D4F,4445,3A20,203C,2072,6574
906   DATA 7572,6E20,3E20,3A3D,2045,4E44,4520,1B71
907   DATA 0000,0000,049C,0000,04F4,0000,054C,0000
908   DATA 11D0,0000,1164,0000,1472,0000,124C,0000
909   DATA 0254,0000,04C8,0000,0520,0000,058A,0000
910   DATA 120E,0000,1164,0000,1472,0000,124C,0000
911   DATA 0254,0000,317A,0000,3185,0000,318F,0000
912   DATA 32B6,0000,32C5,0000,32CD,0000,32D5,0000
913   DATA 32E0,2053,6563,2F54,7261,633A,2030,3920
914   DATA 0020,5245,4144,2020,0020,5752,4954,4520
915   DATA 0020,4544,4954,2054,722E,2000,2042,4143
916   DATA 4B20,001B,7020,2054,5241,434B,204D,4F44
917   DATA 4520,201B,7100,1B70,2020,5452,4143,4B20
918   DATA 5749,5448,2053,594E,4353,204D,4F44,4520
```

```
919    DATA 1B71,001B,7020,2053,6563,746F,723A,2000
920    DATA 201B,7100,1B70,2057,7269,7465,2074,6869
921    DATA 7320,5472,6163,6B20,746F,201B,7100,1B70
922    DATA 203C,2079,6573,2F6E,6F20,3E20,1B71,0000
923    DATA 0000,049C,0000,04F4,0000,054C,0000,17A4
924    DATA 0000,184A,0000,0254,0000,04C8,0000,0520
925    DATA 0000,058A,0000,17A4,0000,184A,0000,0254
926    DATA 0000,317A,0000,3185,0000,318F,0000,339C
927    DATA 0000,33AE,0000,32E0,2052,4541,4420,5749
928    DATA 5448,2053,594E,4353,2000,2041,4444,522E
929    DATA 2046,4945,4C44,2000,0000,049C,0000,1AAC
930    DATA 0000,1CBE,0000,1B2E,0000,1BCC,0000,19BC
931    DATA 0000,1E2C,0000,0254,0000,04C8,0000,1A28
932    DATA 0000,1CBE,0000,1B2E,0000,1BCC,0000,19BC
933    DATA 0000,1E2C,0000,0254,0000,317A,0000,341C
934    DATA 0000,31A8,0000,342B,0000,31B1,0000,3442
935    DATA 0000,3434,0000,31C4,2043,4C55,5354,3A20
936    DATA 3030,3030,2020,0020,204E,4558,5420,2000
937    DATA 2053,5441,5254,6F66,4649,4C45,2000,2045
938    DATA 4449,5420,001B,7020,2043,4C55,5354,4552
939    DATA 204D,4F44,4520,201B,7100,1B70,2020,5768
940    DATA 656E,206C,6561,7669,6E67,2043,4C55,5354
941    DATA 4552,204D,4F44,452C,206C,6173,7420,7265
942    DATA 6164,2043,6C75,7374,6572,2069,7320,7570
943    DATA 6461,7465,7420,696E,2053,4543,544F,5220
944    DATA 4D65,6E75,6520,201B,7100,1B70,2020,5468
945    DATA 6973,2077,6173,2074,6865,206C,6173,7420
946    DATA 4375,7374,6572,2020,1B71,001B,7020,2046
947    DATA 696C,656E,616D,653A,2020,2020,2020,2020
948    DATA 2020,2046,696C,6561,7474,7269,6275,743A
949    DATA 2020,2020,2020,5374,6172,7463,6C75,7374
950    DATA 6572,3A20,2020,204E,756D,6265,7220,6F66
951    DATA 2042,7974,6573,3A20,201B,7100,1B70,2020
952    DATA 5374,6172,742D,436C,7573,7465,7220,6D69
953    DATA 7420,3C52,4554,5552,4E3E,2069,6E73,204D
954    DATA 656E,7565,2081,6265,726E,6568,6D65,6E2C
955    DATA 206C,6573,656E,2064,7572,6368,203C,7570
956    DATA 3E2C,203C,646F,776E,3E2E,201B,7100,1B70
957    DATA 2057,7269,7465,2074,6869,7320,436C,7573
958    DATA 7465,7220,746F,3A20,1B71,0020,2042,7974
```

```
959   DATA 6573,2070,6572,2053,6563,746F,723A,2000
960   DATA 2020,5365,6374,6F72,2070,6572,2043,6C75
961   DATA 7374,6572,3A20,0020,2042,7974,6573,2070
962   DATA 6572,2043,6C75,7374,6572,3A20,0020,2053
963   DATA 6563,746F,7220,7065,7220,4469,7265,6374
964   DATA 6F72,793A,2000,2020,5365,6374,6F72,2070
965   DATA 6572,2046,4154,3A20,0020,2053,656B,746F
966   DATA 726E,756D,6265,7220,7365,636F,6E64,2046
967   DATA 4154,3A00,2020,5365,6374,6F72,206F,6620
968   DATA 6669,7273,7420,4461,7465,636C,7573,7465
969   DATA 723A,0020,204E,756D,6265,7220,6F66,2063
970   DATA 6C75,7374,6572,733A,2000,2020,4E75,6D62
971   DATA 6572,206F,6620,7369,6465,733A,2000,1B70
972   DATA 2020,4669,7273,7420,4469,7265,6374,6F72
973   DATA 792D,7365,6B74,6F72,206F,6E20,5369,6465
974   DATA 3A20,3020,2054,7261,636B,3A20,3120,2053
975   DATA 6563,746F,723A,2033,2020,1B71,001B,7020
976   DATA 2046,6972,7374,2044,6972,6563,746F,7279
977   DATA 2D73,656B,746F,7220,6F6E,2053,6964,653A
978   DATA 2031,2020,5472,6163,6B3A,2030,2020,5365
979   DATA 6374,6F72,3A20,3320,201B,7100,2020,5375
980   DATA 6264,6972,6563,746F,7279,2020,0020,2052
981   DATA 6561,642F,5772,6974,6520,2020,2000,2020
982   DATA 5265,6164,206F,6E6C,7920,2020,2020,0020
983   DATA 2048,4944,4445,4E20,4669,6C65,2020,2000
984   DATA 2020,4465,6C65,7465,6420,2020,2020,2020
985   DATA 0020,2044,6973,6B65,7474,656E,6E61,6D65
986   DATA 2000,0000,049C,0000,04F4,0000,054C,0000
987   DATA 11D0,0000,2340,0000,2582,0000,040C,0000
988   DATA 0254,0000,04C8,0000,0520,0000,058A,0000
989   DATA 120E,0000,2340,0000,2582,0000,040C,0000
990   DATA 0254,0000,317A,0000,3185,0000,318F,0000
991   DATA 32B6,0000,37A6,0000,37AF,0000,37B9,0000
992   DATA 37C1,2046,4F52,4D41,5420,0020,5846,4F52
993   DATA 4D41,5420,0020,2047,4150,5320,0020,2042
994   DATA 4143,4B20,2000,1B70,2020,466F,726D,6174
995   DATA 2054,7261,636B,204D,6F64,6520,201B,7100
996   DATA 1B70,2020,5472,6163,6B3A,0020,666F,726D
997   DATA 6174,6965,7265,6E20,3F20,203C,7965,732F
998   DATA 6E6F,3E20,201B,7100,1B70,2020,4E6F,7420
```

```
999   DATA 666F,726D,6174,7465,6420,2020,2020,203C
1000  DATA 5461,7374,653E,201B,7100,2020,6F6E,2053
1001  DATA 6964,653A,0020,206F,6620,4472,6976,653A
1002  DATA 001B,7020,5265,616C,6C79,2066,6F72,6D61
1003  DATA 7420,7769,7468,206E,6577,2047,4150,6073
1004  DATA 2062,6574,7765,656E,2054,7261,636B,7320
1005  DATA 616E,6420,5365,6374,6F72,733F,203C,7965
1006  DATA 732F,6E6F,3E20,1B71,001B,7020,5761,6974
1007  DATA 2061,2073,6563,6F6E,642C,2074,6865,6E20
1008  DATA 7072,6573,7320,6B65,7920,1B71,0000,2020
1009  DATA 466F,726D,6174,2054,7261,636B,2020,0000
1010  DATA 0000,0000,049C,0000,0D78,0000,0DF0,0000
1011  DATA 0E68,0000,0E7C,0000,0254,0000,04C8,0000
1012  DATA 0DB4,0000,0E2C,0000,0E68,0000,0E7C,0000
1013  DATA 0254,0000,317A,0000,390E,0000,391E,0000
1014  DATA 392F,0000,393E,0000,394B,2020,4D41,5854
1015  DATA 5241,434B,3A20,3739,2000,2020,4D41,5853
1016  DATA 4543,544F,523A,2030,3920,0020,2049,4E49
1017  DATA 5420,4452,4956,4520,2000,2020,5348,4F57
1018  DATA 2042,5042,2020,0020,2042,4143,4B20,2000
1019  DATA 1B70,2020,494E,4954,2044,5249,5645,204D
1020  DATA 454E,5545,2020,1B71,001B,7020,2042,696F
1021  DATA 7320,5061,7261,6D65,7465,7220,426C,6F63
1022  DATA 6B20,6F66,2061,6374,6976,6520,6472,6976
1023  DATA 6520,2020,203C,2070,7265,7373,206B,6579
1024  DATA 203E,2020,1B71,001B,7020,2020,4469,7265
1025  DATA 6374,6F72,7920,7374,6172,7473,2061,7420
1026  DATA 5369,6465,3A20,3020,5472,6163,6B3A,2031
1027  DATA 2053,6563,746F,723A,2033,2020,1B71,001B
1028  DATA 7020,2020,4469,7265,6374,6F72,7920,7374
1029  DATA 6172,7473,2061,7420,5369,6465,3A20,3120
1030  DATA 5472,6163,6B3A,2030,2053,6563,746F,723A
1031  DATA 2033,2020,1B71,0000,0002,0000,0000,0000
1032  DATA 0000,0003,0000,0000,0000,263C,0000,266A
1033  DATA 0000,2698,0000,26C6,0000,26F6,0000,2894
1034  DATA 0000,03C2,0000,2772,0000,27A0,0000,27CE
1035  DATA 0000,27FC,0000,282C,0000,294E,0000,03C2
1036  DATA 0000,3A80,0000,3A8B,0000,3A96,0000,3AA1
1037  DATA 0000,3AAC,0000,3AB8,0000,3AC9,2047,4150
1038  DATA 313A,2036,3020,0020,4741,5032,3A20,3132
```

```
1039  DATA 2000,2047,4150,333A,2032,3220,0020,4741
1040  DATA 5034,3A20,3430,2000,2047,4150,353A,2036
1041  DATA 3634,2000,2042,7974,652F,5365,6B3A,2030
1042  DATA 3531,3220,0020,4241,434B,2000,1B70,2020
1043  DATA 4472,6976,6520,466F,726D,6174,204D,6F64
1044  DATA 6520,201B,7100,1B70,2020,4368,616E,6765
1045  DATA 2047,6170,7320,6265,7477,6565,6E20,5365
1046  DATA 6B74,6F72,7320,201B,7100,1B70,2020,5761
1047  DATA 6974,2061,2073,6563,6F6E,642C,2074,6865
1048  DATA 6E20,7072,6573,7320,6B65,7920,1B71,001B
1049  DATA 7020,2053,4543,544F,5220,4D4F,4445,2020
1050  DATA 1B71,0000,0200,003C,000C,0016,0028,0298
1051  DATA 1B70,2054,7261,636B,3A20,2020,5369,6465
1052  DATA 3A20,2053,656B,746F,723A,2020,4279,7465
1053  DATA 733A,2020,4368,6563,7375,6D28,6865,7829
1054  DATA 201B,7100,1B4A,001B,4B00,1B70,001B,7100
1055  DATA 1B59,2121,001B,4800,1B45,001B,4100,1B42
1056  DATA 001B,4C00,1B6C,001B,7700,1B66,001B,6500
1057  DATA 2020,2020,2020,2020,2020,2000,1B4A,0000
1058  DATA 0000,3C38,0000,3C4C,0000,3C73,0000,3C7C
1059  DATA 0000,3C85,0000,3C8F,0000,3C99,0000,3CB7
1060  DATA 0000,3CC1,0000,3CD7,0000,3CE1,0000,3CEB
1061  DATA 0000,3CF5,0000,3D00,0000,3D20,0000,3D2B
1062  DATA 0000,3D36,0000,3D40,0000,3D4B,0000,3D56
1063  DATA 0000,3D6C,0000,3D77,0000,3D82,0000,3D8D
1064  DATA 0000,3D98,0000,3DA3,0000,3DAE,0000,3DB9
1065  DATA 0000,3DC4,1B70,204E,4F20,424F,4F54,5345
1066  DATA 4354,4F52,201B,7100,1B70,2044,6972,6563
1067  DATA 746F,7279,2D53,6563,746F,7273,2064,6566
1068  DATA 6563,7420,2020,3C6B,6579,3E20,1B71,0020
1069  DATA 6665,686C,6572,3300,2066,6568,6C65,7234
1070  DATA 0020,6665,686C,6572,3520,0020,6665,686C
1071  DATA 6572,3620,001B,7020,4E6F,2044,6973,6B20
1072  DATA 2F20,4E6F,2073,7563,6820,5472,6163,6B20
1073  DATA 1B71,0020,6665,686C,6572,3820,001B,7020
1074  DATA 4E6F,2073,7563,6820,5365,6374,6F72,2021
1075  DATA 1B71,0020,6665,686C,6572,3130,0020,6665
1076  DATA 686C,6572,3131,0020,6665,686C,6572,3132
1077  DATA 0020,6665,686C,6572,3133,2000,1B70,2020
1078  DATA 4469,736B,2069,7320,7772,6974,6570,726F
```

```
1079 DATA 7465,6374,6564,2E20,201B,7100,2066,6568
1080 DATA 6C65,7231,3520,0020,6665,686C,6572,3136
1081 DATA 2000,2066,6568,6C65,7231,3700,2066,6568
1082 DATA 6C65,7231,3820,0020,6665,686C,6572,3139
1083 DATA 2000,1B70,204E,6F20,6D6F,7265,2043,6C75
1084 DATA 7374,6572,201B,7100,2066,6568,6C65,7232
1085 DATA 3120,0020,6665,686C,6572,3232,2000,2066
1086 DATA 6568,6C65,7232,3320,0020,6665,686C,6572
1087 DATA 3234,2000,2066,6568,6C65,7232,3520,0020
1088 DATA 6665,686C,6572,3236,2000,2066,6568,6C65
1089 DATA 7232,3720,0020,6665,686C,6572,3238,2000
1090 DATA 2066,6568,6C65,7232,3920,0000,FFFF,0000
1091 DATA 0074,0808,080C,0808,0808,0808,0806,0410
1092 DATA 080A,0C08,0A0C,080A,5404,0A04,0A0A,0804
1093 DATA 0C08,080C,1006,1208,2028,0A06,0406,0406
1094 DATA 0410,040A,0A06,0406,040E,100A,0604,0604
1095 DATA 0604,0E10,0406,0406,0406,040A,0A0E,180A
1096 DATA 0604,0604,0604,0A14,040A,0A06,0406,040E
1097 DATA 1004,0A0A,0604,0604,0E10,0A06,0406,0406
1098 DATA 040E,0C06,100A,0C10,060A,0C16,0A0C,160A
1099 DATA 0C06,1010,0C0C,1006,100C,0C06,1010,0C0C
1100 DATA 1006,100C,0C16,0606,0608,240E,0604,0808
1101 DATA 0808,060E,0808,1406,044A,0C0A,0A0A,0A0A
1102 DATA 0608,0A0A,0A0A,0C08,0616,1E08,040C,3014
1103 DATA 0810,0C08,0E0E,080A,0808,0604,1212,1C04
1104 DATA 0808,1008,0E0E,080A,1006,100A,060C,480E
1105 DATA 0612,0E06,160E,060A,0A06,040A,0408,0C0A
1106 DATA 0A06,040A,0408,0C16,0E0E,0A0A,0604,0A04
1107 DATA 080E,0A06,040A,0408,0A14,0816,080A,0A16
1108 DATA 1E14,0606,0608,2214,1C06,1C06,0606,0606
1109 DATA 0606,0606,0608,0606,0624,060C,0624,1410
1110 DATA 0C0C,1410,0C0C,1410,0C0C,1410,0C22,080A
1111 DATA 0C08,080A,0E0A,080A,0E0A,080A,0E0A,080A
1112 DATA 0E0A,080A,0E0A,080A,0E0A,080A,0E0A,080A
1113 DATA 0E0A,080A,0E0A,080A,060C,1E3E,263C,4424
1114 DATA 0606,0606,0A08,2E10,0610,100C,0C10,0610
1115 DATA 0C0E,0808,0606,1406,0408,0808,0A1A,1E06
1116 DATA 0408,0808,080A,160C,0850,0E0A,060C,0808
1117 DATA 0A0A,0A12,0610,0A06,0C08,080A,0A0A,1C0E
1118 DATA 0E12,1E06,060A,0822,321A,640E,104A,062C
```

```
1119  DATA 0606,0A32,200A,0A06,1618,1A3A,362E,0C48
1120  DATA 0604,0808,0808,200A,A812,723E,080A,0C08
1121  DATA 0604,163E,1006,1012,120C,2E08,0C10,1212
1122  DATA 0C0C,0608,120A,200C,0622,180E,080A,0A16
1123  DATA 161E,0608,0608,0626,241E,0608,060E,0C1A
1124  DATA 0E0C,0616,0610,220A,0610,0C06,100C,0A08
1125  DATA 0806,0408,0810,080A,2208,0A0C,0A08,0612
1126  DATA 044A,0E0E,0818,0408,060C,0C08,1E0C,0E10
1127  DATA 0E08,1C0E,0C0C,0632,1006,1608,0A14,0806
1128  DATA 0408,0608,060C,1E06,040A,0622,0E06,4210
1129  DATA 1010,0616,080A,0E08,0806,4010,0E10,1216
1130  DATA 180E,2C0C,2C0C,0C0C,0C0C,180A,0A0A,0A0A
1131  DATA 0A22,1E06,0606,0822,1A06,060E,1E08,0808
1132  DATA 340C,1A10,0C08,060A,187A,622C,2008,0E0C
1133  DATA 0C08,0E0C,0C08,0E0C,0C0A,0E0C,2C12,1012
1134  DATA 0C1C,080E,0C0C,080E,0C0C,080E,0C0C,0A0E
1135  DATA 0C2C,0E10,120C,0C1E,0808,080E,0808,080E
1136  DATA 0808,080E,0808,0806,0C1E,0808,080E,0808
1137  DATA 080E,0808,080E,0808,0806,0C24,202C,1A22
1138  DATA 1A12,1414,1414,0C0E,1E1E,1E14,1416,0E12
1139  DATA 080A,0624,0620,0608,040C,0808,080C,0604
1140  DATA 1606,0414,0810,080E,6C04,080A,040C,080C
1141  DATA 0614,0408,1204,0C0C,0C0C,0C0C,1E0C,0C0E
1142  DATA 080C,080A,080C,0C0C,044A,2030,082C,0E42
1143  DATA 0404,0404,0404,0404,0404,0404,04F0,0404
1144  DATA 0404,0404,0404,0404,0404,0404,0404,0404
1145  DATA 0404,0404,04E0,0404,0404,0404,0404,0404
1146  DATA 0404,0404,0404,0404,0404,0404,04A2,0404
1147  DATA 0404,0404,0404,0404,0404,0404,0404,0424
1148  DATA 0404,0404,0404,0404,0404,0404,0404,0404
1149  DATA 0404,0404,0404,0401,0101,3404,0404,0404
1150  DATA 0404,0404,0404,0404,0404,0404,0404,0404
1151  DATA 0404,0126,0404,0404,0404,0404,0404,0404
1152  DATA 0404,0404,0401,2404,0404,0404,0404,0404
1153  DATA 0404,0404,0404,0404,0404,0401,4A04,0404
1154  DATA 0404,0404,0404,0404,0404,0404,0404,0404
1155  DATA 0404,0404,0404,0404,0400
```

# Anhang II - ASCII-Tabelle

Die nun folgende Tabelle stellt alle auf dem ATARI ST darstellbaren ASCII-Zeichen in Form einer Tabelle dar. Den entsprechenden Zahlenwert (ASCII-Wert, zu ermitteln durch die ASC("")-Funktion in BASIC) erhalten Sie, indem Sie die Sedezimal-Ziffer am oberen Rand und diejenige des linken Randes zusammensetzen. So ist z.B. der ASCII-Wert des Buchstabens 'A' $41.

Diese Tabelle wurde mit dem folgenden GfA-BASIC-Programm erstellt, im Anschluß daran finden Sie die Tabelle.

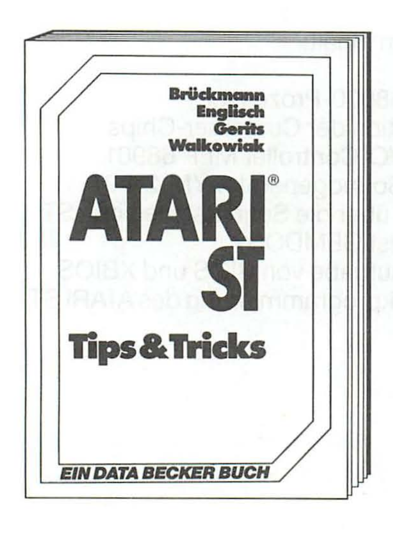
Hier nun das Programm, mit dem Sie sich diese Tabelle auch selbst ausdrucken können (Hardcopy):

```
Cls
For I=0 To 15
  Print At(I*3+7,3);Hex$(I)
  Print At(4,I+4);Hex$(I)
  Deftext 1,0,0,13
  For J=0 To 15
    Text I*24+50,J*16+61,Chr$(I*16+J)
  Next J
Next I
Deftext 1,0,0,4
Text 92,56,"SPC"
For I=1 To 18
  Draw 18,I*16+15 To 424,I*16+15
  Draw I*24-7,31 To I*24-7,303
Next I
Draw 17,31 To 40,47
Repeat
Until Mousek
Edit
```

| / | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | ⇧ | ⇩ | ⇨ | ⇦ | ☒ | ▣ | ☐ | ✓ | 🕓 | 🔔 | ♪ | ␌ | ␍ | ◢ | ◣ |
| 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ə | ESC | [illegible] | [illegible] | [illegible] | [illegible] |
| 2 | SPC | ! | " | # | $ | % | & | ' | ( | ) | * | + | , | - | . | / |
| 3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 4 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 5 | P | Q | R | S | T | U | V | W | X | Y | Z | [ | \ | ] | ^ | _ |
| 6 | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 7 | p | q | r | s | t | u | v | w | x | y | z | { | \| | } | ~ | △ |
| 8 | Ç | ü | é | â | ä | à | å | ç | ê | ë | è | ï | î | ì | Ä | Å |
| 9 | É | æ | Æ | ô | ö | ò | û | ù | ÿ | Ö | Ü | ¢ | £ | ¥ | ß | ƒ |
| A | á | í | ó | ú | ñ | Ñ | ª | º | ¿ | ⌐ | ¬ | ½ | ¼ | ¡ | « | » |
| B | ã | õ | Ø | ø | œ | Œ | À | Ã | Õ | ¨ | ´ | † | ¶ | © | ® | ™ |
| C | ĳ | Ĳ | א | ב | ג | ד | ה | ו | ז | ח | ט | י | כ | ל | מ | נ |
| D | ס | ע | פ | צ | ק | ר | ש | ת | ן | ך | ם | ף | ץ | § | ∧ | ∞ |
| E | α | β | Γ | π | Σ | σ | µ | τ | Φ | Θ | Ω | δ | ∮ | ϕ | ∈ | ∩ |
| F | ≡ | ± | ≥ | ≤ | ⌠ | ⌡ | ÷ | ≈ | ° | • | · | √ | ⁿ | ² | ³ | ¯ |

# Anhang III - Stichwortverzeichnis

Eine riesige Fundgrube wirkungsvoller Tips & Tricks rund um den neuen ATARI ST. Alle Programme sind gut erklärt und können in eigene Anwendungen eingebaut werden. Diese Routinen sind wirklich absolut neu.




Aus dem Inhalt:

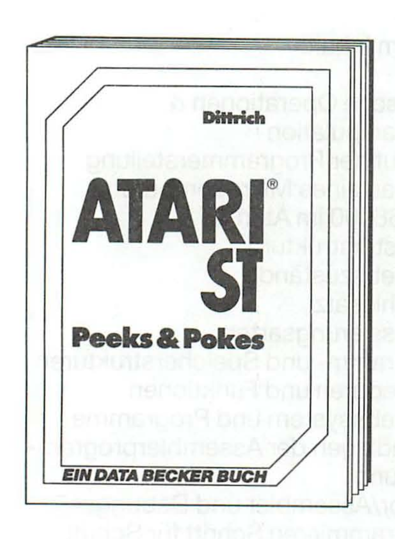
- BASIC und GEM
- Der VDISYS-Befehl
- BASIC und Maschinensprache
- Automatische Hardcopy
- Druckertreiber für EPSON-Drucker
- RAM-Disk für ATARI ST
- Druckerspooler
- Automatisches Starten von TOS-Anwendungen
- C und Maschinensprache
- Hardcopy in Farbe
- GEM intern
- GEM-Anwendungen
- CP/M, Funktionsweise und Aufbau
- CP/M-Emulatoren

und vieles mehr

**Brückmann, Englisch, Gerits, Walkowiak**
**ATARI ST Tips & Tricks**
**Hardcover, 362 Seiten, DM 49,–**
***ISBN 3-89011-118-1***

## Bücher zum ATARI ST

Schlagen Sie dem Betriebssystem Ihres ATARI ST ein Schnippchen. Wie? Mit PEEKS & POKES natürlich! Dieses Buch erklärt leichtverständlich den Umgang mit einer riesigen Anzahl wichtiger POKES und ihren Anwendungsmöglichkeiten. Nebenbei wird der interne Aufbau Ihres neuen ATARI ST prima erklärt.




Aus dem Inhalt:

- Ein-Blick in den ATARI ST
- Innere Konfiguration und Schnittstellen
- Die intelligente Tastatur
- Die Maus als Malstift
- Die internen Speicher
- Zeiger und Stacks
- Diskettenhandling
- Computer-1×1
- Das TOS
- GEM
- Interpreter/Compiler
- Programmiersprachen
- Ein-/Ausgaben

**Dittrich**
**Peeks & Pokes zum ATARI ST**
**198 Seiten, DM 29,–**
***ISBN 3-89011-148-3***



Absender der Zahlkarte
DM 29, Pf — für Postscheckkonto Nr. 789-436

Für Vermerke des Absenders

**Empfängerabschnitt**

Postscheckkonto Nr. des Absenders
DM 29, Pf —
für Postscheckkonto Nr. 789-436
**Lieferanschrift** und **Absender** der Zahlkarte
PLZ Ort
Verwendungszweck
Diskette zum Buch
376 132

**Zahlkarte/Postüberweisung**

PSchA Postscheckkonto Nr. des Absenders Postscheckteilnehmer
Die **stark umrandeten Felder** sind nur auszufüllen, wenn ein **Postscheckkontoinhaber** das Formblatt als **Postüberweisung** verwendet (Erläuter. s. Rücks.)
DM 29, Pf —
(DM-Betrag in Buchstaben wiederholen)
Neunundzwanzig
für **DATA BECKER**
Merowingerstraße 30
in 4000 Düsseldorf
Postscheckkonto Nr. 789-436
Postscheckamt Essen
Ausstellungsdatum Unterschrift

**Einlieferungsschein/Lastschriftzettel**

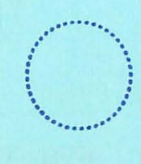
Postscheckkonto Nr. des Absenders
DM 29, Pf —
für Postscheckkonto Nr. 789-436
Postscheckamt Essen
für **DATA BECKER**
Merowingerstraße 30
in 4000 Düsseldorf

# LADEN, STARTEN – KLAR!

Für alle diejenigen, die sich fleißiges Abtippen ersparen und trotzdem alle Programme nutzen wollen, gibt es einen Ausweg:

Mit der nebenstehenden Post-Zahlkarte einfach die Diskette zum Buch bestellen!

Diskette zum Buch „Atari ST Floppy und Harddisk"
DM 29,–



Diskette zum Buch
„Atari ST Floppy und Harddisk"
376 132

Für Mitteilungen an den Empfänger

**Hinweis für Postgirokontoinhaber**
Dieses Formblatt können Sie auch als Postüberweisung benutzen, wenn Sie die stark umrandeten Felder zusätzlich ausfüllen. Die Wiederholung des Betrages in Buchstaben ist dann nicht erforderlich. Ihren Absender (mit Postleitzahl) brauchen Sie nur auf dem linken Abschnitt anzugeben.

1. Abkürzung für den Namen Ihres Postgiroamts (PGiroA) siehe unten
2. Im Feld »Postgiroteilnehmer« genügt Ihre Namensangabe
3. Die Unterschrift muß mit der beim Postgiroamt hinterlegten Unterschriftsprobe übereinstimmen
4. Bei Einsendung an das Postgiroamt bitte den Lastschriftzettel nach hinten umschlagen

**Abkürzungen für die Ortsnamen der PGiroÄ:**

| | | | |
|---|---|---|---|
| BlnW | = Berlin West | Kln | = Köln |
| Dtmd | = Dortmund | Lshfn | = Ludwigshafen am Rhein |
| Esn | = Essen | Mchn | = München |
| Ffm | = Frankfurt am Main | Nbg | = Nürnberg |
| Hmb | = Hamburg | Sbr | = Saarbrücken |
| Han | = Hannover | Stgt | = Stuttgart |
| Klrh | = Karlsruhe | | |

Bedienen Sie sich der Vorteile eines eigenen Postgirokontos

Auskunft hierüber erteilt jedes Postamt

Feld für postdienstliche Zwecke

Einlieferungsschein/Lastschriftzettel
(nicht zu Mitteilungen an den Empfänger benutzen)

Gebühr für die Zahlkarte
(wird bei der Einlieferung bar erhoben)

bis 10 DM --- 90 Pf
über 10 DM (unbeschränkt) 1,50 DM

Bei Verwendung als Postüberweisung gebührenfrei

***DAS STEHT DRIN:***

Darauf haben die ATARI ST-Freaks gewartet: eine umfassende Beschreibung und Analyse der Floppy und Harddisk des ATARI ST. Dieses Buch zeigt Ihnen den richtigen Umgang mit diesen Geräten – von Anfang an. Ganz nebenbei lernen Sie einige Kniffe, die selbst Profis staunen lassen.

Aus dem Inhalt:

- Boot-Sektoren und BIOS-Parameter-Block
- Programmformat auf Diskette und Festplatte
- Filestrukturen
- RAM-Disk und RAM-Disk-Kopierprogramm
- Schnittstelle BASIC/TOS
- beliebige Diskettenformatierung
- Zugriff auf Disk-Controller von BASIC

***UND GESCHRIEBEN HABEN DIESES BUCH:***

Eines der hochkarätigsten Teams von ATARI ST-Spezialisten in Deutschland. Uwe Braun, Software-Entwickler, 68000-Assemblerspezialist, Stefan Dittrich, Informatikstudent, Harddiskspezialist, und Axel Schramm, Fernmeldetechniker, Floppy-Spezialist.

***ISBN 3-89011-132-7***